Neural Pattern Classifying Systems

Theory and experiments with trainable pattern classifiers

PROEFSCHRIFT

Ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus, prof. ir. K.F. Wakker,
in het openbaar te verdedigen ten overstaan van een commissie
aangewezen door het College van Dekanen
op 18 januari 1994 te 14:00 uur

door

Wouter Frederik Schmidt

geboren te Saint-Cloud, Frankrijk,
natuurkundig ingenieur.

Dit proefschrift is goedgekeurd door de promotor prof. dr. I.T. Young.

Dr. ir. R.P.W. Duin heeft als toegevoegd promotor in hoge mate bijgedragen aan het tot stand komen van dit proefschrift.

Cover illustration:
A schematic diagram of the back-propagation training procedure.

This thesis was typeset using MS-Word, MacDraw-Pro and Cricket-Graph III on a Macintosh LC II computer. The camera-ready output was obtained from an Apple LaserWriter Pro 630 printer.

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Schmidt, Wouter Frederik

Neural pattern classifying systems: theory and
experiments with trainable pattern classifiers / Wouter
Frederik Schmidt. [S.1. : s.n.]. - Ill.
Thesis Technische Universiteit Delft. - With ref. - With
summary in Dutch.
ISBN 90-9006716-7
NUGI 811/815
Subject headings: pattern recognition / neural networks

Aan Liesbeth.
Aan mijn ouders.

Contents

Summary

In this thesis the most important development in *neuro-engineering*, the *multi-layer perceptron*, is analyzed from a *statistical pattern recognition* point of view. The multi-layer perceptron fits well into the general model of a *statistical pattern classifier* and has therefore been chosen as the subject of research. A large number of applications are reported using the multi-layer perceptron for pattern recognition purposes and it has become an interesting question how these systems compare to traditional techniques.

The field of pattern recognition has been dominated by a mathematical and statistical approach and the classical techniques are therefore inspired by statistical or mathematical arguments. In neuro-engineering the human brain and its neurons are the source of inspiration and this field of research leads to different models. The arguments that lead to the multi-layer perceptron are explained in this thesis and these systems are clearly different.

The multi-layer perceptron is analyzed and compared on *theoretical* and *practical* aspects. The theoretical aspects are inspired by the issues that have shown to be important in statistical pattern recognition. The following topics are discussed in this thesis:

- The *capabilities* or *complexity* of the multi-layer perceptron.
- The problem of estimating the parameters of the multi-layer perceptron.
- The asymptotic (*infinite*) sample size properties of the multi-layer perceptron.
- The *finite* sample size properties of the multi-layer perceptron.
- The relation between classifiers complexity and the expected probability of error.
- The selection of the best classifier from a number of alternative classifiers.

The applicability of multi-layer perceptrons is investigated by comparing this classifier with two classical pattern classifiers for seven real-world problems. The nearest-mean and nearest-neighbor classifier are used in this comparison because these classifiers also reveal information about the structure of the data. In the comparison the nearest-neighbor classifier and sometimes also the nearest-mean classifier are superior to the classifier implemented by the multi-layer perceptron. This result is pessimistic because it is difficult to compare these classifiers on a fair basis, due to the large number of design issues that have to be optimized for the multi-layer perceptron.

It has not become completely clear if it has a real advantage to use the multi-layer perceptron especially if the accuracy of this classifier is concerned. These systems seem to be, however, less critical with respect to feature extraction because this issue is ignored in most papers that describe applications of the multi-layer perceptron. The hidden units, found in the multi-layer perceptron, are shown to perform a kind of

feature extraction. This observation probably explains why feature extraction is less critical and this is surely an advantage of the multi-layer perceptron.

It is shown in this thesis that the theory and knowledge of small sample problems needed to be refined. A simple comparison of the number of parameters to the number of samples used to train this classifier, is shown to be a pessimistic rule-of-thumb for these new classifiers. This leads to the problem how to quantify or define a classifier's capacity if the number of parameters is not appropriate. An approach inspired by earlier work is proposed as alternative to a definition known as the Vapnik-Chervonenkis dimension.

List of Global Symbols

Symbol	Use
$\|\{\cdot\}\|$	Number of elements in a set $\{\cdot\}$.
α	Momentum term in back propagation learning.
$\alpha(\mathbf{x})$	Decision function, returns the estimated label $\hat{\omega}_i$ for given pattern $\mathbf{x}$.
β	Regularization term.
c	Number of different classes or categories.
d	Number of features or pattern dimensionality.
D	Set of design patterns.
$\|D\|$	Number of patterns in the design set.
δ	Mahalanobis distance.
Δ_y	The delta corresponding to pattern $\mathbf{y}$ (see section 4·3·1).
$\Delta_{y,j}$	The delta of the j-th unit in the next layer corresponding to pattern $\mathbf{y}$.
$e_H(\mathbf{x},\mathbf{x}')$	*Hamming* distance between pattern $\mathbf{x}$ and $\mathbf{x}'$.
$e_1(\mathbf{x},\mathbf{x}')$	An improved distance measure used for NETtalk.
ε	Probability of error.
ε^*	Minimal probability of error or Bayes error.
ε_m	Error of a classifier trained with a particular set of m samples.
$\hat{\varepsilon}_r$	Re-substitution estimate of the probability of error.
$\hat{\varepsilon}_t$	Estimate of the probability of error using an independent test set.
$\varepsilon(g)$	The true probability of error of a specific classifier g().
$\hat{\varepsilon}(g)$	The estimated probability of error of a specific classifier g().
$E[\cdot]$	Expectation operator.
f()	A monotonic increasing function.
$g_i()$	A discriminant function.
G	A set or family of classifiers.
h	Number of units in a hidden layer.
$\mathbf{H}$	Approximation of the Hessian inverse matrix.
η	Learning rate or correction increment.
$\mathbf{j}$	Search direction in conjugate gradient descent.
J()	A criterion function.
k	Number of neighbors.
$\overline{L}$	Expected or average loss, the cost related to a specific classifier.
$L(\hat{\omega}_i \mid \mathbf{x})$	Average conditional loss associated with decision $\hat{\omega}_i$, given pattern $\mathbf{x}$.
$l(\hat{\omega}_i \mid \hat{\omega}_j)$	Costs associated with decision $\hat{\omega}_i$, if ω_j is the correct classification.
λ	Eigenvalue of a matrix or system.
Λ	Diagonal matrix with eigenvalues on diagonal.

m	Number of samples.
$\mathbf{M}$	A matrix containing all the weights of a layer of output units.
$\mathbf{M}_i$	Weight vector of one unit i of a unit in an output layer.
$\mathbf{M}_{i,j}$	Component j of weight vector of output unit.
n	Number of considered distinct classifiers.
N	Number of trainable parameters in a feed forward network.
$p(\mathbf{x})$	Probability density.
$p(\mathbf{x} \mid \omega_i)$	Conditional probability density of pattern $\mathbf{x}$ given class ω_i.
$P(\omega_i \mid \mathbf{x})$	*A posteriori* probability of class ω_i given pattern $\mathbf{x}$.
$P(\omega_i)$	*A priori* probability of an object of class ω_i.
$\mathbf{Q}$	Matrix with columns that are eigenvectors of the original matrix.
ρ	Correlation between J_{mse} and $\hat{\varepsilon}_t$.
$\Re^d$	The d dimensional Euclidean space.
$\mathbf{S}$	Covariance matrix.
$S_G(m)$	The growth function of a set of classifiers G.
t	A desired response (target) for a system.
$t_{\mathbf{y}}$	The desired response of a system for pattern $\mathbf{y}$.
$\mathbf{t}$	A desired response vector of a system.
$\mathbf{t}_{\mathbf{y}}$	A desired response vector of a system for pattern $\mathbf{y}$.
T	Set of independent test patterns.
$\|T\|$	Number of patterns in the test set.
V	Capacity of a set or family of classifiers G.
$\mathbf{w}$	An augmented weight vector.
$\mathbf{w}^{(i)}$	The augmented weight vector at iteration i.
$\tilde{\mathbf{w}}$	A scaled weight vector, not necessarily to unit length.
$\mathbf{W}$	A matrix of weight vectors.
$\mathbf{W}_i$	Weight vector i.
$\mathbf{W}_{i,j}$	Component j of weight vector i.
ω_i	A specific class or category.
$\hat{\omega}_i$.	The estimated class or decision from a pattern classifier.
$\mathbf{x}$	A pattern or feature vector.
ξ	A random pattern from the design set D.
$\bar{\mathbf{x}}$	Expected value or mean value of $\mathbf{x}$.
$\bar{\mathbf{x}}_{\omega_i}$	Expected value or mean value of $\mathbf{x}$, for class ω_i.
$\mathbf{x}_i$	A individual component i of a feature vector.
$\mathbf{x}_{\omega_i}$	A prototype pattern vector for class ω_i.
$\mathbf{y}$	Augmented pattern vector.
$\tilde{\mathbf{y}}$	Class normalized and augmented pattern vector.

1 Introduction

1·1 Developments in pattern recognition

At this moment anyone can see that the human brain is superior to digital computers at many tasks. Only for tasks that are based on simple arithmetic, does the computer outperform the brain. This observation, together with the notion that the switching logic in current computer technology is many orders of magnitude faster than the refractory period of biological neurons, motivated researchers to gain insight into the principles of biological computation [Hertz 1991, chapter 1]. The following properties of the human brain serve as desirable properties for artificial systems:

- Robust and fault tolerance: The nerve cells in the brain are known to die every day and human performance does not seem to suffer from this. The currently developed processors become so complex and contain so many transistors, that fault tolerance or robustness against malfunctioning transistors, becomes relevant for these systems.
- Flexibility: The current systems can not easily adjust to a new environment and require reprogramming. The human brain adapts itself to new situations and *learns* by experience.
- Noisy data: Humans can deal with information that is inconsistent, noisy or inherently probabilistic from nature. The current *intelligent* systems, like rule-based or expert systems, are based on applying strict rules that generate difficulties with inconsistent or noisy data.
- Anticipation: The human brain can handle unforeseen situations, apply knowledge from other domains, and try to extrapolate this to new circumstances.
- Large amounts of data: The human system can deal with large amounts of input data and quickly extracts the relevant properties from that data.
- Parallel: The human brain is highly parallel and with slower devices (the neurons) outperforms their counterparts in digital computers, the transistors.

The field of *neuro-engineering* tries to create or enhance current systems such that they obtain capabilities of the human brain (see also section 2·5). The previous list points out some properties of the brain that are often discussed in the literature and it serves as motivation for a large number of researchers active in this field.

It is interesting to compare the goals of neuro-engineering with similar goals formulated for the field of *pattern recognition*. Since the introduction of digital computers there has been a constant effort to expand the domain of applications, to build or program a machine that can do things that have been never done before. It was noticed in those early years that the ability of machines to *monitor* their environment and to react accordingly was very limited.

The apparent ease with which vertebrates and even simple organisms such as insects perform *perceptual* tasks, is encouraging and frustrating at once. One of the goals of pattern recognition and scene analysis is to duplicate these perceptual tasks with a computer. An important aspect in perception is pattern classification, the assignment of a physical object or event to one or more specified categories [Duda 1973, chapter 1].

The motivations for research in pattern recognition and neuro-engineering are both based on the observation that machines build by humans, can not perform tasks or operate in a way that biological systems seem to do very easily. The goal of neuro-engineering is even more ambitious than pattern recognition. The latter only focuses on a specific aspect, that is in my opinion ambitious enough, namely the ability to recognize objects or events from the environment.

If this list of desirable properties of the human brain is evaluated, a number of them apply to pattern recognition systems. This is not surprising due to the large overlap in motivation between the disciplines pattern recognition and neuro-engineering. In pattern recognition, systems exist that *learn* from data given by an expert or teacher and adapt to a changing environment. These methods are called *supervised learning* (see section 2·2·4) and show (a limited) flexibility.

One approach to pattern recognition is called *statistical pattern recognition* and uses decision theory from statistics to create classifying systems. These systems can handle noisy and inconsistent data due to their probabilistic approach and have shown to be able to handle new (unseen) situations. This last aspect is called *generalization* in pattern recognition and is an important issue in this field.

Finally in pattern recognition *unsupervised* techniques are developed that find structure in the data themselves thereby extracting the relevant properties. Other methods are specially designed to reduce the amount of data and preserve the important features present in the data as much as possible. The methods are referred to as *feature extraction*.

The conclusion from this comparison is that besides robustness and parallelism a number of desired properties of biological systems can be found in pattern classifying systems. An obvious question that becomes apparent now is whether all systems designed by neuro-engineers are pattern recognition systems. The answer is that this is partially true; about 80% of the work in this field can be classified as pattern recognition (see section 2·5).

In the past seven years the field of neuro-engineering or *neural networks* has seen an explosive growth. Although not all developments in this field are related to pattern

recognition, it has surely given a boost to pattern recognition and the application of pattern recognition methods to real world problems. A number of new developments and approaches to pattern recognition can be found in neural network literature and this relatively new field is therefore interesting for researchers from pattern recognition.

1·2 A new model for pattern recognition

Lacking a complete theory about perception one possible approach is based on statistical decision theory (section 2·3). The perception problem is transformed in such a way that this decision theory can be applied. This means that the recognition problem is reduced to a set of numbers and these values are used to build a pattern recognition system. The statistics of these numbers are important and, when these are known, then an abstract mathematical model of the system easily follows.

This approach is straightforward unless the statistics of the numbers, measured to classify an object, are unknown. There exist classifiers that assign an object in a statistical way without explicitly calculating probabilities. Another approach is to simply assume a mathematical model for the pattern recognition system and adjust a number of parameters in that model such that the classifier works as desired (see section 2·4).

The strict mathematical approach to pattern recognition has traditionally guided the choice of an assumed model when the statistical distributions were unknown. In mathematical terms a linear or quadratic function is the obvious choice. This is probably inspired by knowledge that any continuous function can be approximated by a Taylor series and a first-order or second-order approximation is often sufficient. The Gaussian hypothesis coupled to the central limit theorem also leads to either a linear or a quadratic model. A linear classification model is mostly preferred because it contains the least number of trainable parameters, a design issue that is important in pattern recognition.

In neuro-engineering the inspiration of a model for a pattern recognizer is obtained from biological systems and results in different proposed models. The most simple model from neuro-engineering, the *perceptron*, is the linear classifier. The limitations of a linear classifier are obvious from a mathematical point of view and after a long and almost silent period (1970-1985), a non-linear and biologically inspired model emerged from this research field. This development, known as the *multi-layer perceptron*, initiated the explosive growth of neural network's research.

The choice of the multi-layer perceptron as pattern classifier is, in my opinion, a development that is associated to the field of neuro-engineering. A linear or second-order function is an obvious mathematical choice but the proposed multi-layer perceptron does not seem to be logical in that sense. The use of the multi-layer

perceptron has no theoretical motivation, except that it has a weak relation [Crick 1989] to the neuronal structure of the brain.

When, however, no knowledge about the statistical structure of the recognition problem is known, no preference can be given to any proposed model, so the multi-layer perceptron should be considered as an alternative. The strong point of this model is the large number of applications that are reported in literature, where the multi-layer perceptron has been successfully applied. A weak point of these publications is that the performance is generally not compared with traditional models. The conclusion that can be drawn so far is that multi-layer perceptrons can solve pattern recognition problems. It is not known if they can perform this task better than traditional methods.

1·3 Scope of this thesis

The scope of this thesis is to analyse the developments in neuro-engineering from a pattern recognition point of view and to investigate the applicability of newly developed systems for solving recognition tasks. In my opinion most neural network researchers are not aware of the knowledge acquired in pattern recognition, about building machines that can perform perceptual tasks. On the other hand many researchers active in pattern recognition are ignorant of the achievements in neural networks. A number of publications in neural networks indicate that some pattern recognition theories need to be refined.

In *chapter 2* the traditional pattern recognition approach is discussed, specially statistical pattern recognition. Practical aspects have shown a need to limit the applicability of pattern recognition and these design issues are discussed in detail. The consequences of these limitations for neural networks is one of the topics of this thesis. At the end of this chapter a short introduction to the field of neural networks is given.

Chapter 3 deals with the simple perceptrons that are the predecessors of the multi-layer perceptrons. The first part of this chapter is concerned with the well-established theory of these simple structures. This theory, although not new, is presented here for a number of reasons. First a far more rigid theory exists for these classifiers than for the multi-layers perceptrons and the *local* behavior of multi-layers perceptrons can approximately be predicted with this theory. A second and related reason is that it can serve as a source of inspiration to analyse the more complex systems. The third motivation for this review is to clarify the reasoning that leads to the creation of the multi-layers perceptrons. The last reason is to present all this theory and knowledge in a unified and comprehensive way.

The simple perceptron is modified with a non-linear output, needed to create networks that are the topic of the next chapter. The consequence of this modification for the accuracy is shown, an observation that is rarely reported in literature. The practical

design considerations are discussed and this chapter is ended by establishing the link to multi-layer perceptrons.

In *chapter 4* the limited theory of multi-layer perceptrons is presented. There are a number of different training methods discussed and these techniques are evaluated as to how effective they are for training this new type of classifier. In the next section the asymptotic properties and the relation of feed forward classifiers to feature extraction are shown. A few recent developments that are interesting for pattern recognition are discussed at the end of this chapter.

In traditional pattern recognition it has been shown that besides asymptotic properties, practical design limitations, finite sample sizes for example, are important for the successful application of a pattern classifier. In *chapter 5* a list of design considerations is given together with pointers to relevant literature. The second part of this chapter shows the influence of these design issues on the performance of the trained classifier. It becomes clear from this chapter that training a near to optimal neural classifier is not very simple.

The complexity of the classifier and the generalization capabilities as a function of the sample size is an important issue in traditional pattern recognition. The conclusion is that the number of free parameters should be reduced as much as possible, certainly if a very limited number of samples is available. In *chapter 6* it is noticed that the number of free parameters in some neural network applications is so large compared to the sample size that the reported generalization results seem unrealistic from a classical point of view. The capacity and generalization properties of feed forward classifier are investigated and it is shown that the classical design guidelines are very pessimistic for neural classifiers.

One of the important applications of neural networks is the mapping of written English text to the phonemes that can be used to synthesize speech. This mapping problem is reformulated as a classification problem and solved by training this classifier with a set of commonly used English words. In *chapter 7* this application is investigated from a statistical pattern recognition point of view. Furthermore six additional data sets are used to compare the classification capabilities of feed forward classifiers to traditional pattern recognition methods.

A problem that arose during my research was the possibility to simulate neural networks. The combination of full flexibility and an efficient use of the computer resource was not offered by public domain or commercial software. Furthermore an interface to statistical pattern recognition routines was an important demand. In *chapter 8* a simulation environment is described that I developed, that is specially designed for research in neural networks and pattern recognition.

Finally in *chapter 9* the results of my research are summarized, together with some directions for future work.

2
Pattern Recognition

2·1 Introduction

In this chapter the general theory of *statistical* pattern recognition, as far as it is needed in this thesis, is presented. The Bayes decision theory, the statistical approach to pattern recognition, is based on the assumption that the decision problem can be posed in probabilistic terms. If these probabilities are known or an infinite amount of data can be acquired to measure these probabilities then the Bayes theory is straightforward to apply.

In most practical situations, however, little is known about the true probabilities and data acquisition can be expensive or sometimes more data can be unavailable. If a finite amount of data is available the application of the Bayes theory becomes complicated. The notion how to design the *best* possible classifier is not clear any more. Methods that use no prior information about underlying densities have therefore received increased attention in recent years ([Jain 1988b] page 213). In particular systems inspired by biological counterparts have resulted in a renewed activity in this part of pattern recognition.

This chapter is meant as theoretical background to the state of art in pattern recognition as established up to approximately 1980. Those topics that are of interest for the analysis of the latest developments will be presented in this chapter. The latest publications, specially from biologically inspired systems or better known as *artificial neural networks*, show some remarkable results and are worth being investigated from a traditional pattern recognition point of view. At the end of this chapter a short introduction to neural networks is given.

2·2 The basic model

2·2·1 Pattern recognition

In one approach to pattern recognition a *fixed* number of relevant measurements, called *features,* are used to make a decision about an object represented by these observations. A set of features is called a *pattern* or *feature vector* and is described by a vector $\mathbf{x}$. The d individual *components*, $\mathbf{x}_1 .. \mathbf{x}_d$, are assumed to be representative and sufficient to recognize the underlying object.

Any device designed for sorting patterns into different categories is called a *pattern classifier*. It is postulated that the number of different categories c is finite and the *different states of nature*, the classes, are denoted by $\omega_1 .. \omega_c$. One class can correspond to a *null* or *reject* category, which implies that not enough evidence is captured in the presented pattern to assign a category.

The basic model of a pattern classifier is a device with d continuous or discrete valued inputs and one discrete output (see figure 2·1). The output signal is restricted to c distinct values. Each value of the response represents one of the categories ω_i to which a pattern can be placed. The labels estimated by a pattern classifier are denoted by $\hat{\omega}_i$.

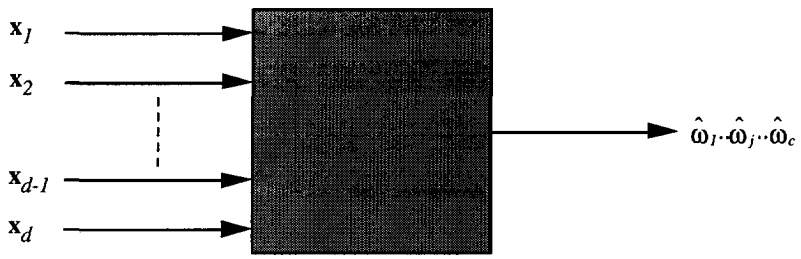

Figure 2·1: The general form of a pattern classifier.

2·2·2 Feature selection

An important issue in designing a pattern classifier is how to select a number of measurements for a given physical situation. Before operating a classifier a decision must be made about the properties that distinguish the different objects from each other. The following problem is how to quantify these properties into measurements.

Take for example a pattern recognition problem where long elongated objects must be distinguished from round compact objects, that are present in a digital image. One or more digital operators must be designed that quantify the differences in shape into numbers (see for example [Bowie 1977] and [Young 1974]). The transformation of the raw data to a more compact and descriptive form is called *feature extraction.*

Unfortunately there is still little theory to guide the selection of a representative and sufficient amount of features. Intuitive ideas and hints from experts play an important role in the design procedure. An important problem, see also section 2·4, is that the optimal number of features is not only dependent on the complexity of the problem, but is also related to the available number of examples used to design the classifier.

It is, however, beyond the scope of this thesis to discuss the problem of feature extraction in detail and in the following it is assumed that a number of features d is

wisely chosen. In chapter 3 of [Young 1986] the problem of feature selection and extraction is discussed.

2·2·3 Decision surfaces in pattern space

A pattern **x** can be represented by a point in a d dimensional Euclidean space $\Re^d$. The coordinates of this point are determined by the values of the measured features and the pattern vector connects the origin with this point. This space is referred to as *pattern space* or *feature space*.

A pattern classifier is a device that maps the points of the pattern space into the different classes. If c classes are present, the pattern classifier will map this space onto labels $\hat{\omega}_i$, that can be represented by category numbers 1 to c. The pattern space is divided into a number of regions, called *decision regions* in pattern recognition. *Decision surfaces* separate the decision regions from each other and play an important role in pattern recognition. It is required that the number of decision surfaces is finite.

There are many ways to mathematically represent a classifier. The most common representation is with a set of *discriminant functions* $g_i(\mathbf{x})$, $i = 1,..c$. These functions are chosen such that a vector **x** is assigned to class ω_i if:

$$g_i(\mathbf{x}) > g_j(\mathbf{x}), \quad \forall i \neq j \tag{2·1}$$

The decision boundaries follow easily from this description. These are given by the following set of equations:

$$g_i(\mathbf{x}) - g_j(\mathbf{x}) = 0, \quad \forall i \neq j \tag{2·2}$$

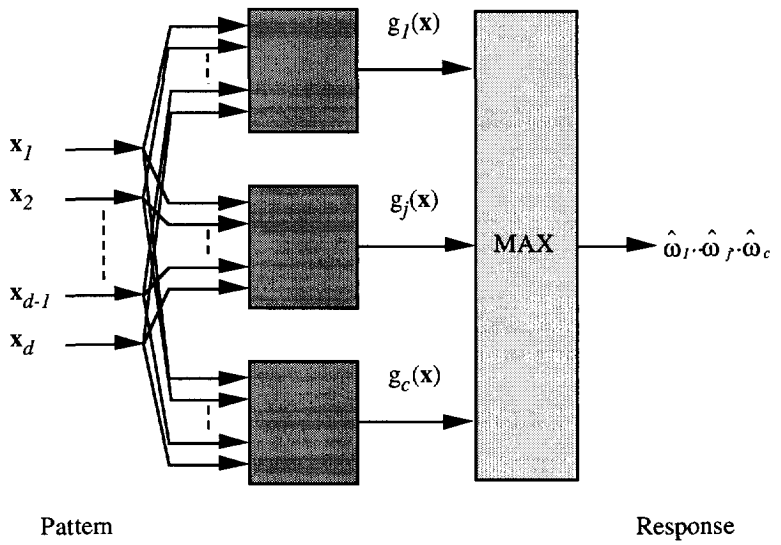

Figure 2·2: A more detailed view of a pattern classifier

An interpretation of the general form of a classifier is given in figure 2·2, which is a more detailed view of figure 2·1. The classifier computes c discriminant functions and selects the category corresponding to the largest discriminant value. Clearly this choice of model is not unique, every $g_i(\mathbf{x})$ can be replaced by $f(g_i(\mathbf{x}))$, where f() is a monotonic increasing function.

A commonly used simplification for the two-category case, is to use one function $g(\mathbf{x})$ instead of two functions $g_1(\mathbf{x})$ and $g_2(\mathbf{x})$. This new function is chosen as:

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) \tag{2·3}$$

The decision rule is modified accordingly, ω_1 is chosen if $g(\mathbf{x}) > 0$ and ω_2 otherwise. It is obvious that a machine constructed this way is consistent with the general multi-category model.

2·2·4 Discriminant functions and training methods

In the literature two methods are distinguished in selecting discriminant functions, the *parametric* and *nonparametric* approach. The main difference between these methods is the amount of available *a priori* knowledge about the pattern recognition task to be performed.

In the parametric case the discriminant functions are chosen based on *a priori* knowledge about the probabilities of the patterns to be classified. It is possible that only the functional forms of these probabilities are known and that these models contain some unknown parameters. The unknown parameters are estimated from a set of examples, known as a *training set* or *training sequence*.

An example of a parametric method, often found in textbooks on pattern recognition, is the assumption that the patterns are generated with normal or Gaussian probabilities. If the mean and covariances of the probabilities are unknown for the different categories, these values can be estimated from the training data using maximum likelihood estimates for these parameters.

Nonparametric methods are applied if no assumptions can be made about characterizing parameters. Some of these techniques bypass any probability estimation in the decision procedure and directly return estimated class memberships from the training set. A good example of such a rule is the nearest neighbor rule (see [Duda 1973, chapter 4]).

Although not always recognized as a nonparametric method, one can just assume a functional form for the discriminant functions. Such a function can be a function normally characterized as a parametric method, however, with no supporting *a priori* knowledge (see [Nilsson 1965, section 1·7]). The discriminant function can for example be chosen to be linear, quadratic or piece wise linear. The unknown parameters can again be estimated by a training procedure using a set of training patterns.

In the previous paragraphs the existence of a training set is assumed for the construction of the classifier. This training sequence is a set of two sequences, the

sequence of feature vectors and a corresponding category identification sequence. A single training sample is the pair of feature vector and corresponding class label. In pattern recognition, methods that require a design set labeled by an expert or supervisor are called *supervised* learning methods.

Unsupervised training methods are methods that do not need a labeled data set. These methods try to find structure in the data themselves, assuming some compactness hypothesis. In this thesis, however, only supervised learning methods are discussed and therefore this topic is not covered here. See for example [Jain 1988a] on this topic.

2·3 Bayes decision theory

2·3·1 Statistical decision theory

The statistical decision theory is the theoretical foundation of statistical pattern recognition. Central to this theory is the specification of a loss function, $l(\hat{\omega}_i \mid \omega_j)$, that specifies the costs when the machine places a pattern belonging to class ω_j accidentally into category $\hat{\omega}_i$. The *average conditional loss* is the expected loss associated with a decision $\hat{\omega}_i$ given an unknown pattern $\mathbf{x}$. This average conditional loss $L(\hat{\omega}_i \mid \mathbf{x})$ can be expressed as:

$$L(\hat{\omega}_i \mid \mathbf{x}) = \sum_{j=1}^{c} l(\hat{\omega}_i \mid \omega_j) P(\omega_j \mid \mathbf{x}) \tag{2·4}$$

A decision rule $\alpha(\mathbf{x})$ is a mathematical formulation of a classifier as shown in figure 2·1. The average loss, related to a specific rule $\alpha(\mathbf{x})$, is the expected value of the average conditional risk. In formulations, where $E[\cdot]$ is the expected value operator, this can be expressed as follows:

$$\mathbf{E}[L(\hat{\omega}_i \mid \mathbf{x})] = \int L(\alpha(\mathbf{x}) \mid \mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{2·5}$$

The decision rule that minimizes equation (2·5) is said to be *optimum* and is also called a *Bayes machine*. A classifier with minimum average loss is a classifier that selects action $\hat{\omega}_i$, for which $L(\hat{\omega}_i \mid \mathbf{x})$ is minimum for a given $\mathbf{x}$.

2·3·2 Two category classification

If we apply the theory of the previous section to a two class problem, only four values for the loss function must be specified. The conditional loss given by equation 2·4 reduces to the following set of two equations:

$$\begin{aligned} L(\hat{\omega}_1 \mid \mathbf{x}) &= l(\hat{\omega}_1 \mid \omega_1) P(\omega_1 \mid \mathbf{x}) + l(\hat{\omega}_1 \mid \omega_2) P(\omega_2 \mid \mathbf{x}) \\ L(\hat{\omega}_2 \mid \mathbf{x}) &= l(\hat{\omega}_2 \mid \omega_1) P(\omega_1 \mid \mathbf{x}) + l(\hat{\omega}_2 \mid \omega_2) P(\omega_2 \mid \mathbf{x}) \end{aligned} \tag{2·6}$$

For the minimum loss or Bayes machine, the pattern classifier should return $\hat{\omega}_1$ if:

$$[l(\hat{\omega}_2 \mid \omega_1) - l(\hat{\omega}_1 \mid \omega_1)]P(\omega_1 \mid \mathbf{x}) \; > \; [l(\hat{\omega}_1 \mid \omega_2) - l(\hat{\omega}_2 \mid \omega_2)]P(\omega_2 \mid \mathbf{x}) \tag{2·7}$$

The well-known Bayes rule:

$$P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})} \tag{2·8}$$

can be combined with 2·7, together with the assumption that errors have higher costs than a correct classification. This leads to the following reformulating of the optimal decision strategy:

$$\frac{p(\mathbf{x} \mid \omega_1)}{p(\mathbf{x} \mid \omega_2)} > \frac{[l(\hat{\omega}_1 \mid \omega_2) - l(\hat{\omega}_2 \mid \omega_2)]}{[l(\hat{\omega}_2 \mid \omega_1) - l(\hat{\omega}_1 \mid \omega_1)]} \frac{P(\omega_2)}{P(\omega_1)} \tag{2·9}$$

The probability density $p(\mathbf{x} \mid \omega_i)$ is called the *likelihood* of ω_i and represents the probability of observation $\mathbf{x}$ for a given class ω_i. The ratio of the likelihood's on the left hand side of equation 2·9 is called the *likelihood ratio*. The right side of equation 2·9 is not a function of $\mathbf{x}$ and can therefore be considered as a constant or a threshold. A Bayes classifier evaluates whether the likelihood ratio exceeds this threshold.

2·3·3 Minimum error classification

The optimal classifier, is a classifier that operates with minimum expected cost, related to the losses $l(\hat{\omega}_i \mid \hat{\omega}_j)$ of accidentally placing an object into the wrong class. A commonly used loss function is the so-called *symmetrical* or *zero-one* loss function.

$$l(\hat{\omega}_i \mid \omega_j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases} \quad \forall i, j = 1,...,c \tag{2·10}$$

This function assigns no loss to correct classification and unit loss to any error. The average loss (2·5) is now equal to the probability of error (notation ε) and the conditional loss of equation 2·4 reduces to the following simple form.

$$L(\hat{\omega}_i \mid \mathbf{x}) = \sum_{i \neq j} l(\hat{\omega}_i \mid \omega_j)P(\omega_j \mid \mathbf{x}) = 1 - P(\omega_i \mid \mathbf{x}) \tag{2·11}$$

The optimal or Bayes classifier is now a classifier with minimal probability of error. The corresponding error of this classifier is called the *Bayes error* and it is noted by ε^*.

2·3·4 Minimal loss and discriminant functions

The Bayesian decision theory leads to a set of discriminant functions $g_i(\mathbf{x})$ that can be expressed using the *a posteriori* probabilities and the problem specific losses $l(\hat{\omega}_i \mid \omega_j)$. Remember that the *maximum* discriminant function specifies the appropriate class and that Bayesian classification requires selecting the class with the *minimum* loss. A Bayesian classifier is now easily represented with a set of discriminant functions as follows:

$$g_i(\mathbf{x}) = -L(\hat{\omega}_i \mid \mathbf{x}) \tag{2·12}$$

For the minimum error rate case, formula 2·11 must be combined with 2·12. Together with the property that all discriminant functions $g_i(\mathbf{x})$ can be replaced by $f(g_i(\mathbf{x}))$, where f() is a monotonic increasing function, the following simplified form can be used:

$$g_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x}) \tag{2·13}$$

2·4 Classifier design in practice

2·4·1 Design and testing

For supervised learning procedures a set with labeled examples is needed to construct the classifier. Furthermore an estimate of the classifier performance is often required; one needs to know if the application of a pattern recognition system leads to acceptable loss in operation. It is well known (see for example [Kanal 1971]) that the error estimated with the set of vectors used for the design is biased, resulting in an optimistic estimate of the true performance. The error estimated on the design set, $\hat{\varepsilon}_r$, is called *apparent error* or *re-substitution error.*

Suppose that only a finite set of examples is available. This set must be used to design and test the pattern recognition procedure. The data used to create the classifier can not be used to evaluate the performance and therefore the total set of examples must be split into two parts. One part of the data set is now used for training (set *D*) and the second part is used for testing (set *T*). The error estimated on an independent test set is, $\hat{\varepsilon}_t$, called *test set error.*

A second important reason to measure the classifier performance is the situation where different classifiers are compared and the best version (see for example [Raudys 1982]) has to be selected from those alternatives. If a finite number of samples are available, an optimal use of the available information must be made. The size of the design set influences the accuracy of the classifier and the size of the independent test set determines the accuracy of the test set error, used for the selection.

The minimal requirement that one would like to meet is that the ranking of the estimates corresponds to the ranking of the true performances (ε). It is well known that the estimate:

$$\hat{\varepsilon}_t = \frac{|\{\mathbf{x} \in T \mid \alpha(\mathbf{x}) \neq \omega_i\}|}{|T|} \tag{2·14}$$

where *T* is the set of test patterns and $|\{\cdot\}|$ denotes the number of elements in the specified set, is a random variable distributed according to the binomial distribution (see [Duda 1973] page 74). The expected value of this random variable is equal to the true probability of error:

$$E[\hat{\varepsilon}_t] = \varepsilon \tag{2·15}$$

The variance of this estimate is equal to:

$$E[(\hat{\varepsilon}_t - \varepsilon)^2] = \frac{\varepsilon(1-\varepsilon)}{|T|} \tag{2·16}$$

It is important to note that the requirement to have estimates of the performance of a set of classifiers that reflect the true ranking of the classifiers, leads to a large number of test patterns if the true performances are close (less than 1%) to each other. See for example figure 3·6 page 75 of [Duda 1973] or figure 2·4 of section 2·4·8 in this thesis.

If a large part of the data is needed to estimate the performance of one or more classifiers, less data is available to train the classifier. A classifier trained with only a few samples will obviously lead to classifiers with less performance. A balance must therefore be found between the number of samples used for training and the number of samples used for testing (see [Jain 1982] or [Kanal 1971]).

A suggested solution to this problem is the *leave-one-out* procedure. If m is the total number of available samples, all possible partitions of size 1 for the test set and m-1 for the training set are evaluated. The estimate of the classification error obtained by taking the average value over all m different classifiers, is an unbiased estimate for a classifier based on m-1 samples. Unfortunately this method is known to have a large variance (see [Ness 1980]) plus the high computation demands that can be involved in training the m classifiers.

Bootstrap techniques try to determine the statistical properties of an estimate when little is known about the underlying distributions. The biased re-substitution error $\hat{\varepsilon}_r$ is corrected by an offset, estimated by employing a bootstrap estimate to the training data (see [Jain 1987]). Bootstrap methods also suffer from a high variance if m is low, although [Jain 1988b] report that reasonable results were obtained by using this method for selecting optimal window sizes for Parzen classifiers.

2·4·2 Infinite sample size and recognition accuracy

Assume that an *infinite* amount of data is available. The question arises then what will happen with the Bayes error rate if the pattern dimension d is increased. In [Waller 1978] it is shown that if one extra feature is added to the patterns and the old data set (I) can be recovered from the new data set (II), that the following holds:

$$\varepsilon_I^* \geq \varepsilon_{II}^* \tag{2·17}$$

This implies that the Bayes error is a monotonic decreasing function of the pattern dimension. In [Laveen 1971] the sufficient condition is given, to ensure that perfect separation can be approached by increasing d. The implication of this condition is that even with variables with very small contributions to discrimination, perfect separation will be possible by using a sufficient number of them. The addition of another variable will never do any worse.

2·4·3 Finite sample size and recognition accuracy

In practical situations the classifier is constructed by replacing the probabilities in equation 2·4 by estimations based on a finite number of examples. In the parametric case for example unknown parameters of the distribution are replaced by their maximum likelihood estimates. A decreasing relationship between pattern dimension and classifier error does not necessarily hold any more.

This is a very uncomfortable situation. There is no longer an obviously natural notion of optimality, whereas this was very clear with the original Bayes rule of equation 2·4. The substitution of the maximum likelihood estimates causes a peaking observed in the performance of many practical classifiers (see for example [Duin 1978], [Jain 1982], [Kanal 1971]). A commonly observed characteristic is that an increase in the pattern size d first decreases the recognition error, but after a critical value the classifier error increases.

A typical graph is shown in figure 2·3. In this figure a linear classifier is trained to distinguish two Gaussian distributed classes with 20 samples available for each class. For both classes the covariance matrix is the unit matrix. The mean vector for class ω_1 is the zero vector and for class ω_2 is a vector with all elements value 0.5. The pattern dimensionality is increased, but the number of samples is fixed. While the Bayes error is monotonic decreasing, the classifier trained with the finite data increases after a specific point.

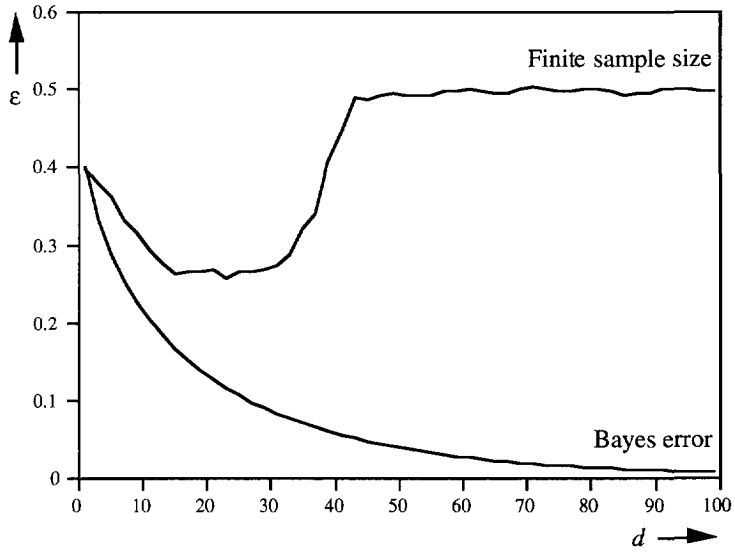

Figure 2·3: Example of the observed classification error of a pattern classifier trained with a finite number of samples versus the pattern dimension d. The classification error is the average value for ten different training sets. The Bayes error is plotted for comparison.

2·4·4 Expected probability of error

In the situation, where only a finite number of samples are available, the design of a pattern classification system has become quite complicated. In this situation the classifier with minimum Bayes risk is not any more the optimal choice, but the classifier trained with only m samples and the minimum expected risk is now the best choice.

Because a training set is only *one* realization of m samples selected from the universe, the probability of misclassification has become a random value also. The expected value of misclassification, regarding all possible realizations of m samples from the universe of discourse, is suggested by [Raudys 1976] as the selection criterion for the optimal classifier. In formulation, select the classifier for which the following holds:

$$\mathbf{E}[\varepsilon_m^i] \leq \mathbf{E}[\varepsilon_m^j] \quad \forall i \neq j \tag{2·18}$$

Here ε_m^i is the probability of misclassification of classifier i trained with only m samples. An important question is what factors influence the value of ε_m. In [Duin 1976] the following expression is derived for a two class classification problem, with equal *a priori* probabilities:

$$\varepsilon^* \leq \mathbf{E}[\varepsilon_m] \leq \varepsilon^* + \frac{1}{2}\mathbf{E}[\int |\hat{p}(\mathbf{x} \mid \omega_1) - p(\mathbf{x} \mid \omega_1)| d\mathbf{x} + \int |\hat{p}(\mathbf{x} \mid \omega_2) - p(\mathbf{x} \mid \omega_2)| d\mathbf{x}] \tag{2·19}$$

From this equation it becomes clear that $\mathbf{E}[\varepsilon_m]$ is dependent on the sample size, sample dimension d, true underlying densities and the parametric form chosen for $\hat{p}(\mathbf{x} \mid \omega_i)$ or the classifier. The parametric form chosen, determines the criterion function that is optimized to find the parameters of a classifier. A different choice results in the optimization of a different criterion. The optimized criterion, together with the method used to find the minimum or maximum of that function, influences the expected probability of misclassification (see for example [Smith 1972]).

2·4·5 Asymptotic error rate expansions

The selection rule suggested in formula 2·18 imposes large difficulties. The complicated dependency of $\mathbf{E}[\varepsilon_m]$ with respect to the underlying distributions, sample size and pattern dimension leads to complicated theoretical analysis of the expected performance. Some asymptotic (Taylor series) approximations are obtained for classification problems involving multivariate Gaussian densities.

In almost all asymptotic error rate expansions two equally probable classes are represented by multivariate Gaussian distribution with a common d by d covariance matrix $\mathbf{S}$ and mean vectors $\bar{\mathbf{x}}_{\omega_1}$ and $\bar{\mathbf{x}}_{\omega_2}$. The decision rule that results in minimum classification error is linear and is given by:

$$g(\mathbf{x}) = \mathbf{x}^T \mathbf{S}^{-1}(\bar{\mathbf{x}}_{\omega_1} - \bar{\mathbf{x}}_{\omega_2}) - \frac{1}{2}(\bar{\mathbf{x}}_{\omega_1} + \bar{\mathbf{x}}_{\omega_2})\mathbf{S}^{-1}(\bar{\mathbf{x}}_{\omega_1} - \bar{\mathbf{x}}_{\omega_2}) \tag{2·20}$$

This discriminant function is optimal if the covariance matrix $\mathbf{S}$ and mean vectors are known. Usually the parameters $\mathbf{S}$, $\bar{\mathbf{x}}_{\omega_1}$ and $\bar{\mathbf{x}}_{\omega_2}$ are estimated from the training data and are replaced in formula 2·20 by their maximum likelihood estimates. The resulting classifier may not be optimal any more and the distribution of $\hat{g}(\mathbf{x})$ is called W-statistics.

At least 9 different asymptotic expressions are published that describe the average probability of error of $\hat{g}(\mathbf{x})$, where the average is taken with respect to all possible realizations of m training samples, [Bowker 1961], [Sitgreaves 1973], [Smith 1972], [Okamoto 1963, 1968], [Anderson 1973], [Efron 1975], [Schervish 1981], [Raudys 1972], [Deev 1972], [Kharin 1984].

The difference in expressions obtained depends on the approximations that are used during the derivation. Most of these expressions are good up to order m^{-1} while few of them are good up to order m^{-2}. In [Wyman 1990] seven different expansions are compared by means of Monte-Carlo simulations. The simple expansion derived by Raudys [Raudys 1972] was found to yield the best results. This expansion ([Wyman 1990] or [Raudys 1980]) is given by:

$$\mathbf{E}[\hat{g}(\mathbf{x})] \approx \Phi(-\frac{1}{2}\delta^2\sqrt{\frac{m-d}{m\delta^2+4d}}) \tag{2·21}$$

Here Φ is the standard normal cumulative distribution and δ is the Mahalanobis distance, defined by:

$$\delta = \sqrt{(\bar{\mathbf{x}}_{\omega_1} - \bar{\mathbf{x}}_{\omega_2})^T \mathbf{S}^{-1} (\bar{\mathbf{x}}_{\omega_1} - \bar{\mathbf{x}}_{\omega_2})} \tag{2·22}$$

Selection formula 2·18 and expansion 2·21 can now be combined to select the optimal number of features, given a fixed amount of m samples. A slightly different approach can be found in [Jain 1978]. In this paper the error expansion of [Sitgreaves 1973] is used to calculate the minimum increase in Mahalanobis distance needed to keep the same error rate when one measurement is added. The minimal increase, dependent on the sample size is given by:

$$\delta_{d+1}^2 - \delta_d^2 \approx \frac{\delta_d^2}{2m-3-d} \tag{2·23}$$

Unfortunately these expansions only hold for the very specific case of Gaussian probabilities with equal covariance structure. If the two classes have unequal covariance matrices, then the estimation of the expected small sample probability of error becomes extremely difficult and according to [Jain 1982] this does not even exists. In [Raudys 1976] a reference to a Russian publication is given, however, the author argues that this expansion is so complicated that empirically calculated tables are faster to compute. In [Raudys 1976], [Raudys 1980], [Boullion 1975] and [Smith 1972] such tables can be found, some of these tables also include other classifiers or linear classifiers optimized with alternative criterion functions.

2·4·6 Sample size and mean accuracy in pattern recognition

The method of Taylor series approximation lacks theoretical elegance, but is a very common analysis tool. In particular the table lookup approach is a typical engineering solution to the peaking phenomena. It is very accurate and for most situations the only alternative ([Smith 1972]).

A first theoretical approach to the peaking phenomenon was published by Hughes ([Hughes 1968]), with very pessimistic results. Ten years later [Campenhout 1978] showed that the peaking effect in the model by Hughes, arises from an improper comparison of statistical models. The possibility that peaking arises in sub-optimal decision rules, as discussed in the previous sections, is not denied.

In [Chandrasekaran 1977] the general conditions are derived, for the case of independent measurements, under which it can be guaranteed that the peaking phenomenon will not occur. They conclude that the requirement that the features be independent is not a *sufficient* condition to avoid peaking, although this was suggested as a possibility in [Duda 1973] page 77.

A number of alternative approaches try to give a guideline for the number of samples, based on totally different considerations. In [Young 1978] five different of these sample size considerations are listed. The most obvious criterion is that the estimate of matrix **S** should be non-singular, leading to minimal requirement that $m>d$. A different approach by [Cover 1965] considers the probability that a random labeling of m points is linearly separable. He observed that this probability rapidly decreases if $2m>d$, suggesting that the number of samples should be chosen at least as such.

Accordingly to [Jain 1982] it is good practice in pattern recognition that the number of training samples should at least be five to ten times the number of measurements. Although this results is not theoretically justified it is widely accepted as a good rule of thumb. The sample size considerations in [Young 1978] support this too.

2·4·7 Selection of a pattern classifier

In the previous small sample size considerations, the optimal pattern dimension was considered. The optimal complexity and type of classifier is, however, also influenced by the sample size. Assume that it is known *a priori* that we deal with a pattern recognition problem where the patterns are distributed with Gaussian probabilities, with *unequal* covariance matrices. The optimal Bayes classifier is a quadratic function and for equally probable classes is given by:

$$g(\mathbf{x}) = (\mathbf{x}-\overline{\mathbf{x}}_{\omega_2})^T \mathbf{S}_2^{-1}(\mathbf{x}-\overline{\mathbf{x}}_{\omega_2}) - (\mathbf{x}-\overline{\mathbf{x}}_{\omega_1})^T \mathbf{S}_1^{-1}(\mathbf{x}-\overline{\mathbf{x}}_{\omega_1}) + \ln\frac{|\mathbf{S}_2|}{|\mathbf{S}_1|} \qquad (2\cdot24)$$

If the different mean vectors and covariance matrices are unknown they have to be estimated from the training set. In [Raudys 1976] figure 2 and [Raudys 1982] figure 3 it is shown that if the sample size is very low, it is better to use classifier 2·20 because the

expected probability of error is lower for this classifier than for 2·24. This seems surprising because the quadratic classifier is known to be the Bayes optimal one. The explanation for this observation is that many more parameters must be estimated for 2·24 than 2·20 with the same number of training samples. If the sample size is increased, at a certain point the linear and quadratic classifier show equal performance. If the sample size is again increased the quadratic classifier will outperform the linear classifier.

If no *a priori* knowledge is available about the recognition problem, the available data must also be used to select a classifier. In [Raudys 1982] such a situation is investigated and here it is found that if the number of classifiers is large compared to the number of test samples, the selection procedure will cause an increase in classification error. The increase in classification error is due to the selection of a sub optimal classifier from the collection of classifiers considered. This situation can occur where the sample size is sufficiently large to design a parametric classification rule and at the same time is insufficient to select the best classifier from a collection of classifiers.

Theoretical understanding of this selection problem is given by [Vapnik 1982] page 142. Remember that the estimated probability of error $\hat{\varepsilon}_t$ is a random variable with binomial distribution. For each estimate $\hat{\varepsilon}_t$ the law of large numbers holds: as the test set size increases to infinity the estimate will converge to the true probability of error. A bound on the deviation from the true probability, using Hoeffdings theorem 1 [Hoeffding 1963, page 15], is given by:

$$P(|\hat{\varepsilon}_t - \varepsilon| > \Delta\varepsilon) \leq 2e^{-2(\Delta\varepsilon)^2 m} \tag{2·25}$$

We are, however, interested in the convergence for n classifiers, a bound on the simultaneous fulfillment of n of these bounds. For the finally selected classifier this bound is roughly:

$$P(|\hat{\varepsilon}_t - \varepsilon| > \Delta\varepsilon) \leq 2ne^{-2(\Delta\varepsilon)^2 m} \tag{2·26}$$

For this last bound it is assumed that all n estimates $\hat{\varepsilon}_t$ are independent from each other, which is a pessimistic assumption because all n classifiers are trained with the same training samples. From this equation we learn that if the number of classifiers n is increased, it becomes likely that the estimated error of one of these classifiers deviates significantly and will therefore be selected as final classifier. This problem becomes more apparent if the number of classifiers is large compared to the number of test samples, as concluded by [Raudys 1982] from experimental results.

The complexity of this analysis dramatically increases if the selection of the *best* classifier is on basis of the re-substitution errors $\hat{\varepsilon}_r$. The bias of this estimate depends on the complexity of the classifier and for more complex classifiers even worse results should be expected.

2·4·8 Pattern recognition and increasing sample size

From the last sections it has become clear that sample size plays an important role in pattern recognition. In this section the applicability of pattern recognition as function of the sample size is summarized. If for example no knowledge about probability distribution exists and no training data is available ($m = 0$), no statistical pattern recognition is possible. One thing one can do is just to assign all incoming samples to one and the same class ω_1. The associated probability of error for such a classification scheme is bounded by $\varepsilon \leq 1 - \min\{P(\omega_i)\}$.

If a small number of samples is available ($m \approx d$), a simple classification scheme will result in a probability of error that is less than $\varepsilon \leq 1 - \max\{P(\omega_i)\}$. The situation can occur that the Bayes classifier is known to be more complex, however, the amount of data is not sufficient to estimate all the parameters of the more complex classifier. A reduction of classifier complexity or pattern dimension will result in classifiers with a lower expected probability of error.

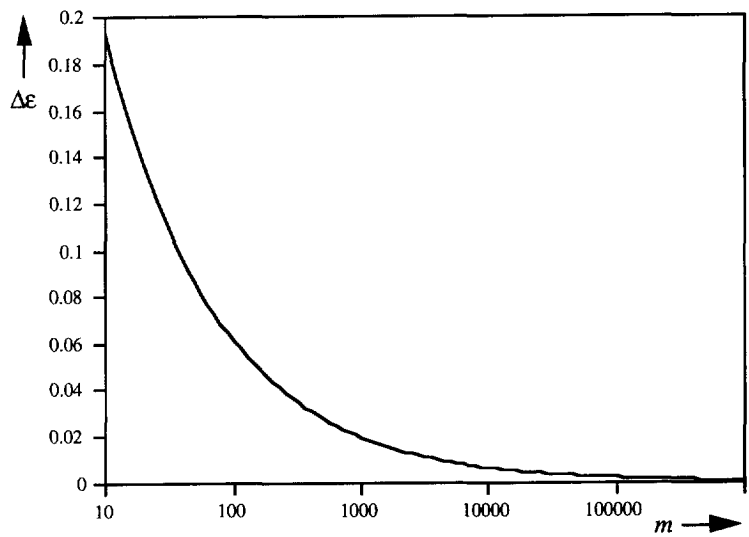

Figure 2·4: Confidence intervals for a true error ε=0.5 and varying size of the test set. For a given size m one can be 95% confident that the observed error is in the interval $0.5 - \Delta\varepsilon \leq \hat{\varepsilon} < 0.5 + \Delta\varepsilon$. Note that for ε<0.5 this interval is smaller.

If again the size of the training set is increased ($m > 5d$), more complex classifiers can be applied. For parametric classifiers near-Bayes results are possible, but the selection of the *best* classifier from a set of n classifier can result in the selection of a classifier with a higher probability of error. The number of possible classifiers n should be limited, due to the finite amount of test samples.

Increasing the number of samples ($m >> 10d$) leads to the comfortable situation that enough design and test samples are available. The optimal parametric or non-parametric

technique can be selected from a set of classifiers without the problem that this procedure will increase the probability of error. It should be noted that this last requirement leads to large test sets, as already mentioned in section 2·4·1, if classifiers are close in performance.

The accuracy of the test sample estimate can now be calculated from equation 2·25 because Hoeffding's inequality is tight (equal sign holds) for the binomial distribution with equal probabilities. In figure 2·4 the 95% confidence intervals are given for varying sizes of the test set. From this graph it becomes clear that large test sets are needed to distinguish two classifiers that are relatively close to each other.

2·5 Artificial neural networks

2·5·1 Introduction to artificial neural networks

Pattern recognition has gained a lot of new attention from a new emerging field, a collection of disciplines referred to as *artificial neural networks* or *connectionist models*. Neuro-engineering is a large and diverse field and therefore it is not easy to give a clear definition of the full domain of research. However, most definitions emphasize the prefix neuro, pertaining the brain. In [Werbos 1991] the following definition is given:

Neuro-engineering is the field of research that tries to copy over the known capabilities of the brain into computer hardware and software.

In the development of brain-like designs, neuro-engineering tries to make use of what is known about how the brain achieves these capabilities. Two different schools of thought can be distinguished.

2·5·2 Subdivisions of neural computation

One school closely follows *only* what is known about the brain, in simplified terms, and looks for emergent computational properties. This group tries to understand specific brain pathways and connections that implement specific abilities, without regarding how these abilities can be learned. An example of such research is the development of a very accurate model of the cochlea of the ear, in order to develop better models of the transfer functions used in adult mammals to preprocess speech data. The implementation of the same transfer function in VLSI, can be used to preprocess speech data ([Lyon 1988]). This group is called by [Werbos 1991] the biologically based or *bottom-up* school.

The other school treats the knowledge of our brain as a loose constraint and focuses on the desired capabilities. Systems are built, making heavy use of statistics, control theory and some of our knowledge about the brain. The key aggregate behavior of interest is the ability to *learn*, from an environment, how to solve complex real world

problems using neural like computations. In [Werbos 1991] this approach is called the *top-down* school of thought.

A more detailed division of this last (top-down) approach, can be distinguished. The *first* sub-group is very much related to statistical pattern recognition and focuses on the learning ability. Three types of learning systems are known, supervised learning, unsupervised learning and *reinforcement* learning. The first two learning paradigms have been known for a long time in pattern recognition (see section 2·2).

Reinforcement learning is similar to supervised learning, however, the actual pattern classes are not known. Only a performance measure of how good a certain response is to a pattern is available. This is similar to teaching a child where to go by saying *colder* or *hotter* as he or she gets closer or further from the goal. Many biologists believe that this is a more accurate model of learning in nature than supervised learning. This approach, although not new to pattern recognition or control theory, is often not distinguished as a different learning method.

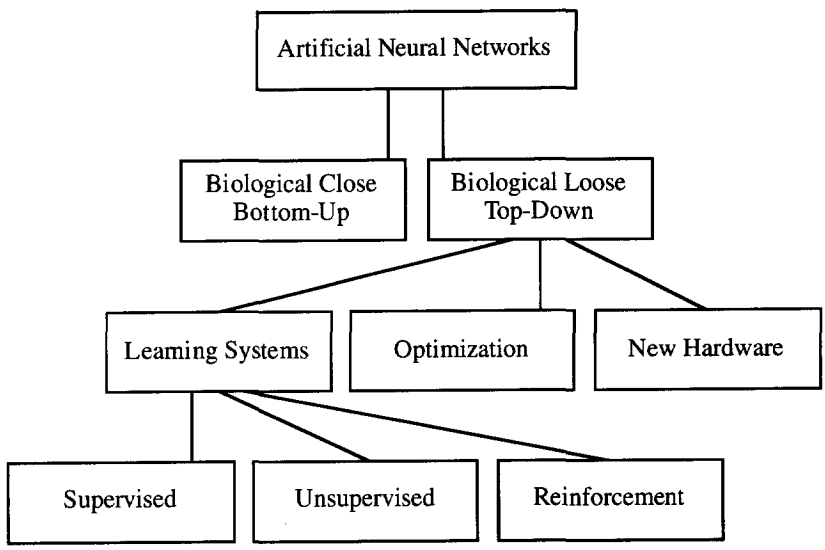

Figure 2·5: Subdivisions of artificial neural networks.

A *second* group tries to apply neural structures to solve complex optimization problems. Work published by [Hopfield 1985] induced great excitement when it was shown that certain artificial neural networks could be used to solve the traveling-salesman problem. Their solution also translates easily to VLSI or optical computers and can be adapted to all kinds of problems where a complex measure of quality must be optimized. Technically these networks are continuous-time, gradient-based, methods for minimizing a user specified energy function.

The last, *third* sub-group, are interested in the human brain as inspiration for a new generation of computers. A million-fold increase in throughput is not likely with current technology in digital electronics and some researchers ([Mead 1989]) believe

that analog, parallel, fixed function VLSI might achieve such an increase. The critics of this analog VLSI argue that this technique is far too restrictive to be useful in any general purpose computer device. The researchers reply that the human brain functions like that and that it seems to be capable of handling a fairly wide range of computational tasks. The challenge is to duplicate the human brain's capabilities for a future generation of computers.

2·5·3 Pattern recognition and neural networks

In figure 2·5 a schematic diagram is given of the subdivisions that can be distinguished in artificial neural networks. This subdivision closely follows the review from [Werbos 1991]. The sheer volume of research makes it almost impossible to classify the full range of paradigms, so figure 2·5 must be regarded as a rough partition.

Werbos guesses that about 80% of the work in this field can be classified as pattern recognition, the learning systems branch of figure 2·5. In most applications of networks to pattern recognition problems, the *back-propagation* method is applied (see chapter 4), a supervised training method. This thesis is mainly focused on this paradigm because it is the most widely used method and very impressive results have been reported with this type of network (see for example [Hertz 1991, pages 130-141]). Roughly 70% of the pattern recognition applications of neural networks use some variation of this method.

Briefly I want to mention two other interesting neural network paradigms. These two are both unsupervised learning methods. The *adaptive resonance theory*, ART, has been developed by Carpenter and Grossberg and shows strong resembles to cluster techniques. A large number of articles by Carpenter and Grossberg have been published on this topic. For an introduction to their theory see [Carpenter 1988] or [Hertz 1991, page 228-232].

A second class of unsupervised methods, also known as *competitive learning*, is published by Kohonen. The Learning Vector Quantization (LVQ), categorizes a given set of vectors into c classes and then represents or classifies each vector into the class in which it falls. The c prototypes divide up the input space into a Voronoi tessellation, similar to the nearest neighbor classifier. An improved version of this algorithm preserves the neighborhood relation as *much* as possible and the resulting mapping is known as *topology-preserving map*. The LVQ method is a type of clustering and the topology-preserving mapping can be regarded as a special type of clustering or projection technique. An introduction to this theory can be found in [Kohonen 1988a] or [Hertz 1991, page 217-244]. A complete description can be found in [Kohonen 1988b].

I agree with [Werbos 1991] that for the *most* part artificial neural networks are technically a subset of statistical pattern recognition, based on a biologically inspired design constraint. A constraint that is a great source of inspiration in the design process. The underlying concepts and paradigms under study are so close and the range of

problems overlap to such a degree, that greater mutual communication can minimize duplication of effort and will enrich both disciplines.

3
Simple Perceptrons

3·1 Introduction

The theory of biologically inspired adaptive pattern recognition systems, as developed until 1969, is presented in this chapter. These systems, known as *perceptrons*, are the predecessors of the multi-layer perceptrons that became so popular in the 1980's. The theory of these simple structures is presented first, because a far more rigid theory exists for this class of trainable pattern classifiers than the multi-layer perceptrons treated in the next chapter.

3·2 The general architecture

3·2·1 Linear discriminant functions

In chapter 2 it was argued that the choice of a proper discriminant function $g_i(\mathbf{x})$ can be very complicated and one often just chooses a convenient functional form. An obvious candidate is the linear discriminant function:

$$g_i(\mathbf{x}) = \mathbf{W}_{i,1}\mathbf{x}_1 + \ldots + \mathbf{W}_{i,j}\mathbf{x}_j + \ldots + \mathbf{W}_{i,d}\mathbf{x}_d + \mathbf{W}_{i,d+1} = \mathbf{W}_i\mathbf{y} \qquad (3\cdot1)$$

In this formula $\mathbf{W}_i$ is a vector containing all free parameters of discriminant function i and vector $\mathbf{y}$ is the augmented pattern vector, obtained from pattern $\mathbf{x}$ by adding a $d+1$st component to this vector with value 1. The substitution of equation 3·1 as a functional block in figure 2·2, results in a family of classifiers that is shown in figure 3·1.

A pattern classifier employing linear functions can simply be implemented using weighting and summing devices and is called a *linear machine*. The training of such a machine can be accomplished by adjusting the values of the parameters $\mathbf{W}_i$, called *weights*, by using a set of training patterns. A number of update-techniques will be discussed in this chapter, but first the decision boundaries of the linear machine are investigated.

3·2·2 Minimum distance classifiers

Before investigating the linear machine, the equivalence with the minimum distance classifier is shown. Assume that for a c class classification problem, c prototype vectors $\mathbf{x}_{\omega_i}$ are given. The squared Euclidean distance between an arbitrary point $\mathbf{x}$ and a prototype is given by:

$$|\mathbf{x}-\mathbf{x}_{\omega_i}|^2=(\mathbf{x}-\mathbf{x}_{\omega_i})(\mathbf{x}-\mathbf{x}_{\omega_i})=\mathbf{x}\cdot\mathbf{x}-2\mathbf{x}\cdot\mathbf{x}_{\omega_i}+\mathbf{x}_{\omega_i}\cdot\mathbf{x}_{\omega_i} \tag{3·2}$$

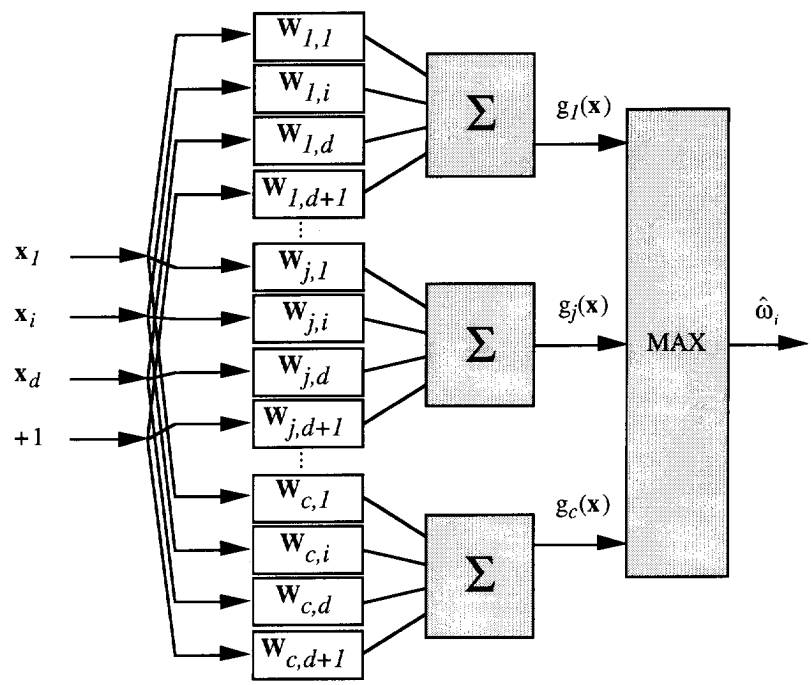

Figure 3·1: A linear classification machine.

The minimum distance classifier places each unknown pattern $\mathbf{x}$ into the category that is associated with the nearest prototype in the squared Euclidean sense. That is, for an unknown pattern all c quantities $|\mathbf{x}-\mathbf{x}_{\omega_i}|^2$ are evaluated and $\mathbf{x}$ is classified to the category with the smallest distance to its prototype. Finding the minimum value of equation 3·2 is equivalent to finding the maximum value for the following expression:

$$g_i(\mathbf{x})=\mathbf{x}\cdot\mathbf{x}_{\omega_i}-1/2\mathbf{x}_{\omega_i}\cdot\mathbf{x}_{\omega_i} \tag{3·3}$$

From this equation it is clear that the minimum distance classifier is equivalent to a linear machine where the first d components of $\mathbf{W}_i$ are given the values of the prototype vector for ω_i and the $d+1$st component is given by the second term in the right hand side of equation 3·3. In formula:

$$\mathbf{W}_i = \begin{pmatrix} \mathbf{x}_{\omega_i} \\ -1/2\,\mathbf{x}_{\omega_i} \cdot \mathbf{x}_{\omega_i} \end{pmatrix} \tag{3·4}$$

3·2·3 Decision surfaces of linear machines

The common boundaries of the decision regions of a linear machine are given by the set of equations of 2·2. For the linear machine these surfaces are linear and given by:

$$(\mathbf{W}_{i,1} - \mathbf{W}_{j,1})\mathbf{x}_1 + \ldots + \ldots + (\mathbf{W}_{i,d} - \mathbf{W}_{j,d})\mathbf{x}_d + \mathbf{W}_{i,d+1} - \mathbf{W}_{k,d+1} = 0 \tag{3·5}$$

There are $c(c\text{-}1)/2$ such equations and thus the same number of surfaces. For $d=2$ a linear surface is called a line; for $d=3$ a plane; and for $d>3$ a hyper plane. The decision surfaces of a linear machine are segments of at most $c(c\text{-}1)/2$ hyper planes. In the special case that the linear machine is a minimum distance classifier, if the weights $\mathbf{W}_i$ can be written in a form given by 3·4, the hyper planes are the perpendicular bisectors of the line segments that join the prototype points.

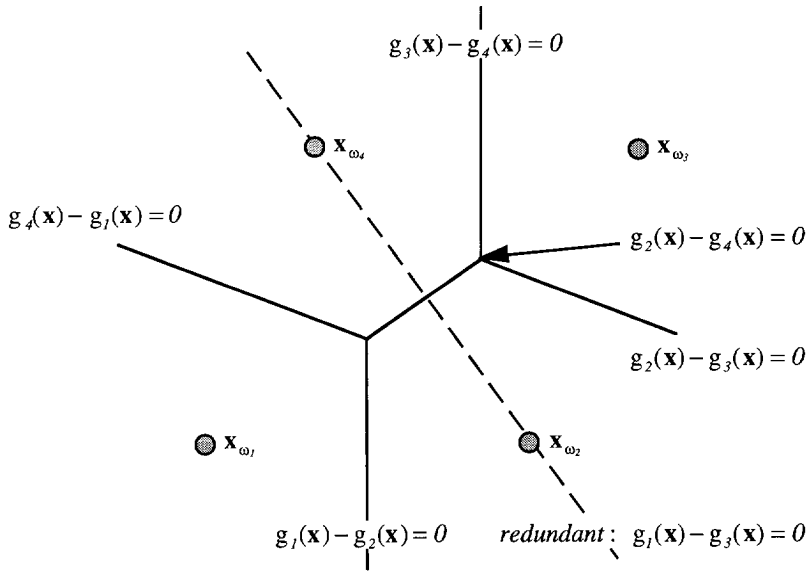

Figure 3·2: An example of the decision regions and surfaces resulting from a minimum distance classifier with four prototype points.

In many cases not all hyper planes defined by equation 3·5 are used in the classification. If two classes ω_i and ω_j are not contiguous, the corresponding hyper plane is unused and is called *redundant*. In figure 3·2 a two-dimensional example is shown of a minimum distance classifier with a redundant hyper plane.

Furthermore it can be shown that the segments of a linear machine always form decision regions that are convex ([Nilsson 1965, page 20] and [Duda 1973, page 133]). Every region must also be singly connected[Duda 1973, page 133], a consequence of

the maximum selector at the output. This tends to make this machine most suitable for problems with unimodal conditional densities $p(\mathbf{x} \mid \omega_i)$ (see [Duda 1973, page 134] for an explanation). This is definitely a limitation of the flexibility of the classifier. An advantage of the linear machine is the analytical simplicity together with the clear interpretation in pattern space of its functionality.

3·2·4 The threshold logic unit

For the two-class case ($c=2$) the simplification given by equation 2·3 reduces the linear machine to one single discriminant function $g(\mathbf{x})$ defined by:

$$g(\mathbf{x}) = \mathbf{w}_1\mathbf{x}_1 + \ldots + \mathbf{w}_j\mathbf{x}_j + \ldots + \mathbf{w}_d\mathbf{x}_d + \mathbf{w}_{d+1} = \mathbf{w} \cdot \mathbf{y} \tag{3·6}$$

If $g(\mathbf{x})>0$, ω_1 should be chosen and ω_2 otherwise. A block diagram of this classifier is shown in figure 3·3 and consists of a summing device, a threshold element and $d+1$ weights. The threshold element is a device that responds with $+1$ if $g(\mathbf{x})>0$ and -1 if $g(\mathbf{x})<=0$ and we must associate categories ω_1 and ω_2 with these responses respectively. This type of classifier is called a *threshold logic unit* (TLU), adaptive linear device (adaline) or α-perceptron.

The interpretation in pattern space of the functionality of the TLU is straightforward. This device separates the pattern space into two regions by a hyper plane given by $g(\mathbf{x})=0$. The mathematics of hyper plane geometry can directly be applied to the TLU and the following properties follow from this theory. The weights $\mathbf{w}_i$ ($i=1..d$) determine the hyper plane orientation, it is the vector normal to the linear surface. The position of the decision surface is proportional to weight element $\mathbf{w}_{d+1}$ and passes through the origin if this element is zero. Finally the distance from the hyper plane to an arbitrary point $\mathbf{x}$ is proportional to the value of $g(\mathbf{x})$.

3·3 Training threshold logic units

3·3·1 Linear separable classes

All algorithms presented in section 3·3 assume that the training data is linearly separable with zero error, except for the procedure described in section 3·3·7. This last algorithm, known as the *pocket* algorithm, tries to work around the severe limitation of perfect linear separability, by proposing an additional test during the training. However, except for this last section zero error linear separability is required in the following sections.

In formal mathematical notation it is obligatory that a solution $\mathbf{w}$ *exists* such that $\mathbf{w}$ is a solution of the set of inequalities:

$$\begin{pmatrix} \mathbf{w}\cdot\mathbf{y} > 0 & \forall \mathbf{y} \in \omega_1 \\ \mathbf{w}\cdot\mathbf{y} < 0 & \forall \mathbf{y} \in \omega_2 \end{pmatrix} \text{ or } \begin{pmatrix} \mathbf{w}\cdot\mathbf{y} > 0 & \forall \mathbf{y} \in \omega_1 \\ \mathbf{w}\cdot -\mathbf{y} > 0 & \forall \mathbf{y} \in \omega_2 \end{pmatrix} \tag{3·7}$$

In this expression $\mathbf{y}$ is again the augmented pattern vector and $\mathbf{w}$ is the vector containing all weights. A pattern for which $\mathbf{w} \cdot \mathbf{y} = 0$ can be arbitrarily classified as either class (a tie) or it can be chosen, as adopted in the following, to call this situation undefined.

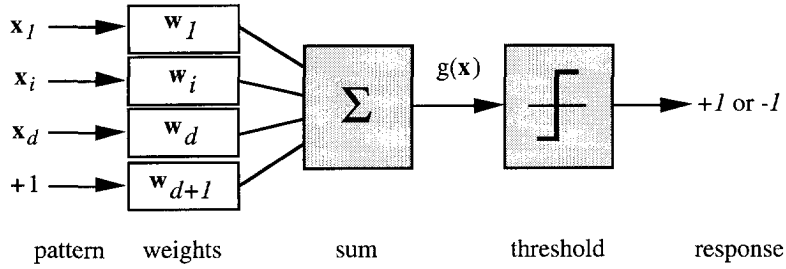

Figure 3·3: The threshold logic unit (TLU).

The alternative formulation of 3·7 suggests a normalization that simplifies the two-category case, the replacement of all samples belonging to class ω_2 by their negatives. With this *class-normalization* of pattern vectors we can forget the labels and look for the vector that is the solution of the set of inequalities:

$$\mathbf{w} \cdot \tilde{\mathbf{y}} > 0 \quad \forall \tilde{\mathbf{y}} \in D \tag{3·8}$$

In this equation $\tilde{\mathbf{y}}$ is the class-normalized and augmented pattern vector and D is the design set. In figure 3·4 an example illustrates this for a linearly separable case in two dimensions. The vectors shown are not the augmented vectors for display convenience. Note that the actual pattern recognition problem does not necessarily have to be linearly separable. The design set D, however, must meet this requirement.

The geometrical interpretation of formula 3·8 is that all augmented and class normalized patterns need to occupy one half of the $d+1$ dimensional space. Using the alternative definition of vector dot product, equation 3·8 can be written in the following form:

$$|\mathbf{w}| \; |\tilde{\mathbf{y}}| \cdot \cos(\varphi_{\mathbf{w}\tilde{\mathbf{y}}}) > 0 \quad \forall \tilde{\mathbf{y}} \in D \tag{3·9}$$

The norm of a vector is always positive so 3·9 can only be fulfilled if a vector $\mathbf{w}$ exists that makes an angle φ with all vectors from the design set that is absolutely smaller than 90 degrees. This is only possible when all vectors occupy one half of the space. Now if we look again at figure 3·4b no positive linear combination of vectors $\tilde{\mathbf{y}}$ can be equal to zero if all these vectors lay in one half plane, except for the trivial case of zero weighting of all vectors. A set of vectors with this property is called *positively linear independent*. So finally linear separability is in the algebraic sense equivalent to positively linear independence.

The requirement of perfect linear separability, sometimes called a *linear dichotomy*, can thus be formulated in four ways: In pattern recognition perspective a linear classifier must exist that can classify the samples with zero error. The geometrical

interpretation leads to the requirement that all augmented and class-normalized patterns must occupy one half plane. In algebraic terms the set of equations given by 3·8 must be solvable. Finally the set of design patterns $\tilde{\mathbf{y}}$ should be positively linear independent.

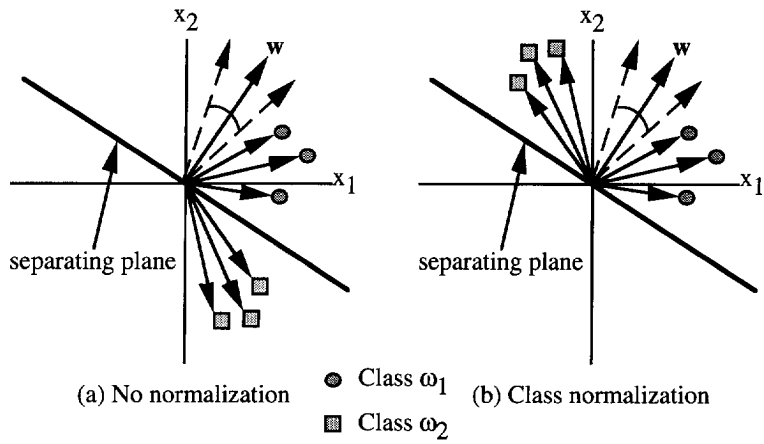

Figure 3·4: Linear separable samples shown as original and class normalized patterns, together with the separating plane and solution region for weight vector.

3·3·2 Stochastic training procedures

Suppose a design set D is a linear dichotomy and that for some particular pattern $\tilde{\mathbf{y}}$ the TLU with weight vector $\mathbf{w}$ has a response that is erroneous. To be more specific, pattern $\tilde{\mathbf{y}}$ is either on the negative side or exactly on the pattern hyper plane determined by $\mathbf{w}$. This error can be rectified by moving the hyper plane such that the erroneous pattern now results in a positive response, or that (at least) the response after the change is less negative.

Such an adjustment can be realized by adding the class-normalized and augmented pattern $\tilde{\mathbf{y}}$ to the weight vector to create a new weight vector $\mathbf{w}'$. That is,

$$\mathbf{w}' = \mathbf{w} + \eta\tilde{\mathbf{y}} \tag{3·10}$$

where η is a positive number called the *correction increment* or *learning rate*. The learning rate controls the extent of the adjustment. For a sufficiently large value of η the correction immediately results in a correct response of the TLU, but for a small learning rate the classifier moves towards a less negative output. If we take the vector dot product with vector $\tilde{\mathbf{y}}$ on both sides of equation 3·10, the following expression is found:

$$\mathbf{w}' \cdot \tilde{\mathbf{y}} = \mathbf{w} \cdot \tilde{\mathbf{y}} + \eta\tilde{\mathbf{y}} \cdot \tilde{\mathbf{y}} \geq \mathbf{w} \cdot \tilde{\mathbf{y}} \tag{3·11}$$

From this equation clearly the proposed adjustment changes the TLU towards a correct classifier. Training methods that are based on this principle, are called *error*

correction procedures. To train a TLU to classify the complete design set correctly, the training patterns are presented one at a time for trial. A trail consists of comparing the actual response with the desired response. If the pattern is correctly classified then the TLU is not updated. For a pattern with an erroneous response, however, the weights are update according to equation 3·11. This learning rule can be formulated as follows:

$$\begin{aligned} &\mathbf{w}\cdot\tilde{\mathbf{y}} > 0 \rightarrow \mathbf{w}' = \mathbf{w} \\ &\mathbf{w}\cdot\tilde{\mathbf{y}} \leq 0 \rightarrow \mathbf{w}' = \mathbf{w} + \eta\tilde{\mathbf{y}} \end{aligned} \tag{3·12}$$

The training patterns may be tried in any order, but depending on the learning rate it can be necessary that patterns must be presented more than one time. The patterns may be presented by cycling through the design set in a fixed order, or by randomly selecting them from the set, as long as the trial of each one recurs. One complete pass of the training set is called a *training sweep* or *epoch* and a cyclic presentation is sometimes called an *iteration*.

The only remaining questions are how to start and stop the training. The initial weights are *not* so important, they can be pre-set to any convenient value (**0** for example) or they can be set at random. Because the design set is required to be a linear dichotomy, the training must be continued by cycling through the patterns, until all patterns are correctly classified. One of the fundamental mathematical results of the theory of error correction procedures is that this is achieved with a finite number of adjustments, if the learning rate is chosen properly. The convergence proof is given in section 3·3·5.

In the remaining part of this section three different choices of η will be discussed. For the *fixed-increment rule*, the learning rate η is taken to be any fixed number greater than zero. A common value is one, so each weight is updated by the addition of the corresponding component of $\tilde{\mathbf{y}}$. A single update does not necessarily result in a proper classification of this vector by the new TLU. A variation on this procedure changes η such that after an update a pattern is classified correctly. The learning rate needed to change the classifier, so that the pattern is moved onto the decision plane is given by:

$$\mathbf{w}'\cdot\tilde{\mathbf{y}} = \mathbf{w}\cdot\tilde{\mathbf{y}} + \eta\tilde{\mathbf{y}}\cdot\tilde{\mathbf{y}} = 0 \Rightarrow \eta = \frac{-\mathbf{w}\cdot\tilde{\mathbf{y}}}{\tilde{\mathbf{y}}\cdot\tilde{\mathbf{y}}} \tag{3·13}$$

The *absolute correction rule* takes a value for the learning parameter that is a fraction larger than the value given by 3·13. In a digital computer the smallest integer greater than this critical value can be chosen. This is the minimum update needed to correctly classify the pattern $\tilde{\mathbf{y}}$. In the *fractional correction rule* a fraction of 3·13 is used as correction increment:

$$\eta = \eta_{fc}\frac{-\mathbf{w}\cdot\tilde{\mathbf{y}}}{\tilde{\mathbf{y}}\cdot\tilde{\mathbf{y}}} \tag{3·14}$$

In this last case the update is proportional to the normal distance of the pattern and decision surface. If $0<\eta_{fc}<1$ the hyper plane is traversed a fraction of this distance towards the presented pattern. If $\eta_{fc}=1$ the plane is moved exactly into the pattern

(3·13) and for $1<\eta_{fc}<2$ the pattern is now a fraction, or the same normal distance for $\eta_{fc}=2$, on the correct side of the plane.

3·3·3 Gradient descent procedures

A second approach to finding a solution for the set of linear inequalities of 3·8 will be to define a criterion function $J(\mathbf{w})$ that has a minimum for a solution vector. The problem of training a perceptron is now a minimization problem, for which a number of methods are known. One method is known as *gradient descent* and is a very simple optimization method to apply. At some arbitrary point $\mathbf{w}$ this method is started and the gradient vector $\nabla J(\mathbf{w})$ is computed at this point. The next value $\mathbf{w}'$ is obtained by moving some distance from $\mathbf{w}$ in the direction of the steepest descent, that is along the negative of the gradient. In formulation this procedure is given by:

$$\mathbf{w}' = \mathbf{w} - \eta \nabla J(\mathbf{w}) \tag{3·15}$$

In this formula η is again a positive factor that sets the step size of the adaptation and is similar to the learning rate found in the previous section. From a pattern recognition point of view the most obvious choice for a criterion function would be the number of misclassified samples, if we have a minimum error classification problem (section 2·3·3). For gradient search this can not be used because this is a piece wise constant function, so no proper gradient search direction can be calculated. The following function is continuous, with a discontinuous first derivative, that is called the *perceptron criterion function*:

$$J_{fi}(\mathbf{w}) = \sum_{\{\tilde{\mathbf{y}} \in D | \mathbf{w}\cdot\tilde{\mathbf{y}} \le 0\}} -\mathbf{w}\cdot\tilde{\mathbf{y}} \tag{3·16}$$

If equation 3·15 is applied to the perceptron criterion function 3·16 the following basic algorithm is found:

$$\mathbf{w}' = \mathbf{w} - \eta \nabla J_{fi}(\mathbf{w}) = \mathbf{w} + \eta \sum_{\{\tilde{\mathbf{y}} \in D | \mathbf{w}\cdot\tilde{\mathbf{y}} \le 0\}} \tilde{\mathbf{y}} \tag{3·17}$$

In words this procedure for finding a solution vector can be formulated as follows. The next weight vector is obtained by adding a fraction of the sum of misclassified samples to the current weight. For this new weight vector the misclassified samples are calculated and if no more samples are misclassified this procedure is stopped, otherwise the weights are updated with the new sum of misclassified samples.

If equations 3·12 and 3·17 are compared, these update rules show a large similarity. In the first update rule a weight is updated for every individual sample that is misclassified. The update rule of equation 3·17 averages all misclassified samples and performs an update with the mean error vector. The update $\eta\tilde{\mathbf{y}}$ is just a noisy estimation of the true gradient, if it is assumed that the training samples are sampled with equal probability from the set D:

$$E[\eta\tilde{\mathbf{y}}] \approx \frac{\eta}{|\{\tilde{\mathbf{y}} \in D \mid \mathbf{w}\cdot\tilde{\mathbf{y}} \le 0\}|} \sum_{\{\tilde{\mathbf{y}}\in D|\mathbf{w}\cdot\tilde{\mathbf{y}}\le 0\}} \tilde{\mathbf{y}} \tag{3·18}$$

Procedure 3·12 therefore approximately optimizes the perceptron criterion function by gradient search. If the value of η is very small, these two methods become almost identical. For the absolute correction and fractional correction rule the corresponding criterion function is given by:

$$J_{fc}(\mathbf{w}) = \frac{1}{2} \sum_{\{\tilde{\mathbf{y}}\in D|\mathbf{w}\cdot\tilde{\mathbf{y}}\le 0\}} \frac{(\mathbf{w}\cdot\tilde{\mathbf{y}})^2}{\tilde{\mathbf{y}}\cdot\tilde{\mathbf{y}}} \tag{3·19}$$

The gradient descent procedure applied to this criterion, results in the following rule:

$$\mathbf{w}' = \mathbf{w} - \eta\nabla J_{fc}(\mathbf{w}) = \mathbf{w} + \eta \sum_{\{\tilde{\mathbf{y}}\in D|\mathbf{w}\cdot\tilde{\mathbf{y}}\le 0\}} \frac{-\mathbf{w}\cdot\tilde{\mathbf{y}}}{\tilde{\mathbf{y}}\cdot\tilde{\mathbf{y}}}\tilde{\mathbf{y}} \tag{3·20}$$

The equations 3·12 and 3·14 combined are again the stochastic approximations of this procedure. The fraction η_{fc} is now just the learning rate η in equation 3·20. It may be clear that the two criterion functions are just two examples. Any function with a minimum for the solution vector and a well-behaved derivative can be used in this approach. In [Duda 1973, page 170] a number of alternatives are discussed; This page lists a summary of descent procedures.

3·3·4 Interpretation of training algorithms

In section 2·2·3 pattern recognition was interpreted in terms of decision surfaces in pattern space. A space similar to the pattern space can be constructed. The weights of the TLU are represented as the coordinates of a point in a $d+1$ dimensional Euclidean space $\Re^{d+1}$. The vector connecting the origin with this point is called the *weight vector* and the space self is called *weight space.*

A weight vector divides pattern space into two regions (see figure 3·5), separated by a hyper plane given by $\mathbf{w}\cdot\tilde{\mathbf{y}} = 0$. The patterns found on one side give a positive response and patterns on the opposite side give a negative response to the given weight vector. In the *weight space* every training pattern divides this space into two regions, again separated by the hyper plane $\mathbf{w}\cdot\tilde{\mathbf{y}} = 0$. Now all weights found on one side of this plane represent TLU's with a positive (correct) response and weights found on the other side conform to negative (incorrect) responding TLU's. The weight space can be considered as the dual space of the pattern space. A solution exists if all the half spaces determined by the patterns have a common overlapping region. This region is always an open convex region, bounded by hyper planes *all* passing through the origin. The resulting region is a convex *polyhedral cone* with the vertex at the origin, as illustrated in figure 3·5.

A training session can be interpreted as a sequential examination of the weight space where one inquires whether the current weight point is on the correct side. If it is not correct the weight point is moved in the direction of the normal of the pattern plane, otherwise the next plane is examined. This process is also illustrated in figure 3·5.

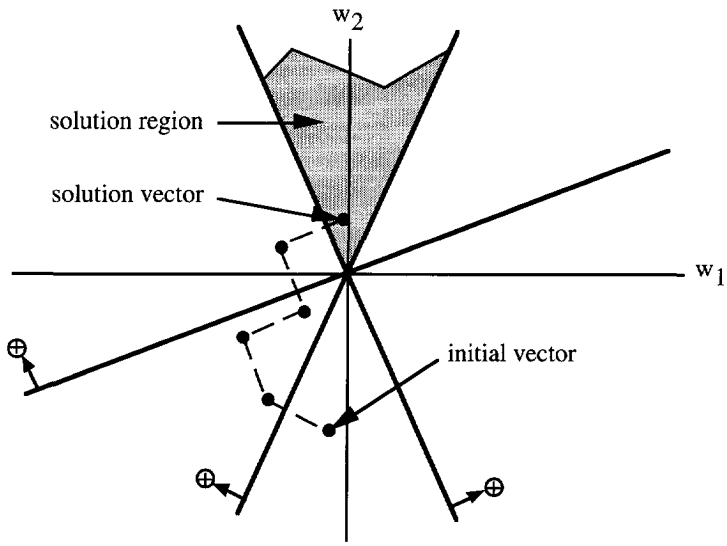

Figure 3·5: A two-dimensional weight space with three pattern hyper planes, together with an example of error-correction training.

3·3·5 Fundamental training theorem

The convergence proof of the perceptron training is shown in this section. This is the central theoretical foundation of the class of iterative training methods discussed so far. The convergence properties shall be investigated for the stochastic training methods, but this theorem obviously holds for the gradient based methods. This can easily be shown by applying equation 3·18.

Furthermore only the fixed increment rule is investigated, but similar proof exists for the absolute correction and fractional correction rules. For these proofs see [Nilsson 1965], [Duda 1973] or [Rosenblatt 1962].

The perceptron convergence theorem: For a linear dichotomy D, the sequence of weight vectors $\mathbf{w}$, $\mathbf{w}'$, $\mathbf{w}''$, ... $\mathbf{w}^{(i)}$ generated by any random or cyclic training sequence drawn from D, using the fixed increment rule and beginning with any initial vector $\mathbf{w}$ will converge after a finite number of updates to a solution vector.

A number of proofs for this theorem exist, two of them can be found in [Nilsson 1965], one in [Duda 1973] and the first proof was given by [Rosenblatt 1962]. The following treats a geometrical argument that shows that it is impossible to apply the fixed increment rule without moving into the solution region. This following proof is a modification of the proofs given by [Nilsson 1965] and [Duda 1973].

The solution region of all weight vectors that satisfy the set of inequalities of equation 3·8, is a polyhedral cone as discussed before. If a solution $\mathbf{w}$ exists, it can

always be scaled to a vector $\tilde{\mathbf{w}}$ such that it is also a solution of the following set of equations:

$$\tilde{\mathbf{w}} \cdot \tilde{\mathbf{y}} > (\eta \underset{\tilde{\mathbf{y}} \in D}{\mathrm{MAX}} \{\tilde{\mathbf{y}} \cdot \tilde{\mathbf{y}}\} + \xi)/2 \quad \forall \tilde{\mathbf{y}} \in D \tag{3·21}$$

Here η is a fixed value chosen for the learning rate and $\zeta > 0$ is any arbitrary constant. The solution region of $\tilde{\mathbf{w}}$ is contained in the solution region of $\mathbf{w}$. The boundaries of this region are the boundaries of the original solution space, shifted by an amount proportional to the right hand side of 3·21. In figure 3·6 the region of the scaled solution vectors is shown.

The proof of the perceptron theorem is shown by computing the decrease in squared distance of the current weight vector, to an arbitrary member of the scaled solution space $\tilde{\mathbf{w}}$, effected by the fixed increment update. This amount is given by:

$$|\tilde{\mathbf{w}} - \mathbf{w}^{(i)}|^2 - |\tilde{\mathbf{w}} - \mathbf{w}^{(i+1)}|^2 = -2\tilde{\mathbf{w}} \cdot \mathbf{w}^{(i)} + \mathbf{w}^{(i)} \cdot \mathbf{w}^{(i)} + 2\tilde{\mathbf{w}} \cdot \mathbf{w}^{(i+1)} - \mathbf{w}^{(i+1)} \cdot \mathbf{w}^{(i+1)} \tag{3·22}$$

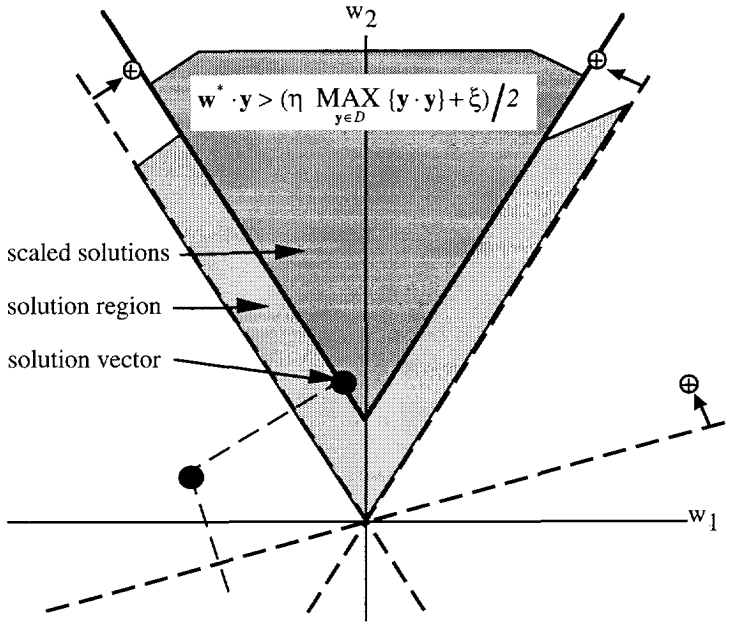

Figure 3·6: The scaled solution region as part of the complete solution region, used for the perceptron convergence proof.

For a correctly classified sample the weights are unchanged so the decrease is zero. For an incorrectly classified sample the weights are updated with the fixed increment rule. The substitution of equation 3·10 into 3·22 results in:

$$|\tilde{\mathbf{w}} - \mathbf{w}^{(i)}|^2 - |\tilde{\mathbf{w}} - \mathbf{w}^{(i+1)}|^2 = \eta(2\tilde{\mathbf{w}} \cdot \tilde{\mathbf{y}} - \eta\tilde{\mathbf{y}} \cdot \tilde{\mathbf{y}}) - \eta\mathbf{w}^{(k)} \cdot \tilde{\mathbf{y}} \tag{3·23}$$

This formula can be bounded, the first term in brackets is bounded by ζ because for vector $\tilde{\mathbf{w}}$ equation 3·21 holds. The second term is greater than or equal to zero, because $\tilde{\mathbf{y}}$ is a misclassified sample. A lower bound for the decrease in squared distance is thus:

$$|\tilde{\mathbf{w}} - \mathbf{w}^{(i)}|^2 - |\tilde{\mathbf{w}} - \mathbf{w}^{(i+1)}|^2 \geq \eta \cdot \zeta > 0 \qquad (3\cdot24)$$

This last inequality shows that the squared distance to some fixed interior point of the solution space, effected by a training step, must exceed a positive amount larger than zero. If each pattern occurs often enough in the training sequence, steps continue to be taken until a weight vector within the solution region is attained, thus proving the convergence. Note that when the training sequence is not balanced and not all training patterns occur often enough then this algorithm is not guaranteed to find a solution.

3·3·6 Training algorithms for multi class machines

A linear machine for more than two classes, as presented in figure 3·1, can be trained with an error correction like procedure. For each class we have to train a discriminant function, as given by equation 3·1. The discriminant function of class ω_i ($g_i(\mathbf{x})$) that corresponds to the applied pattern must give the maximum response to this pattern.

A *generalization* of the error correction procedure can be used to train the linear machine, again provided that a solution exists. The existence of a solution now demands that a set of c vectors $\mathbf{W}_1 .. \mathbf{W}_c$ can be found that can classify a pattern to one of the c classes with zero error. The notion of an augmented pattern $\mathbf{y}$ is still useful, but the class normalization is obviously skipped. Separability in the multi class case requires that all the patterns that belong to one class are separable from the remaining patterns of the design set.

The initial weights are again chosen randomly or set to some prefixed value and weight adjustments are only done if the machine incorrectly classifies a sample. Suppose that a pattern $\mathbf{y}$ is classified to class ω_j but should be classified as ω_i. The vectors $\mathbf{W}_i$ and $\mathbf{W}_j$ are adjusted as follows:

$$\begin{aligned} \mathbf{W}'_i &= \mathbf{W}_i + \eta\mathbf{y} \\ \mathbf{W}'_j &= \mathbf{W}_j - \eta\mathbf{y} \end{aligned} \qquad (3\cdot25)$$

The set of design vectors is cycled through until no more samples are misclassified. The three alternative choices for the training parameter η as well the gradient based approach can be applied. A convergence theorem for the generalized error correction procedure exists and guarantees that the solution will be found in a finite number of steps ([Nilsson 1965, page 87]).

3·3·7 Training non-linear separable classes

The error correction rules form a set of simple methods, combined with good theoretical and geometrical understanding. A limitation is the capabilities of the linear machine. This architecture can implement only convex decision boundaries. A more

severe limitation is, however, the requirement that the design set should be linearly separable. This limits its use to only those *rare* situations that the error rate of the linear discriminant is so low (or zero) that the design set can be assumed to be linearly separable.

If this assumption is unfortunately not true the training will never stop. For these situations, very limited theory exists and only a number of heuristics have been proposed. For example, when cycling through the design set, the weight will visit a number of states and the average vector of those state vectors can be returned as final weight vector.

Another heuristic is to reduce the learning rate η with the number of cycles. If the learning rate is decreased too fast, the weight vector may be far from optimal. On the other hand if the learning rate is decreased too slow the result will depend on the erroneous samples with the largest norm. The decrease in learning can also be a function of the current performance.

The *pocket algorithm* proposed by [Gallant 1990] is an attempt to make the perceptron learning well-behaved for the non-separable case. A proof of the convergence in probabilistic terms is given, although it is limited to tri-state inputs $\{-1, 0, 1\}$ and integral weights. The convergence theorem is based on the fact that for the *fixed increment* method with integer valued inputs, the perceptron learning is a finite state process [Minsky 1969, page 182]. For every reachable weight vector a non-zero probability exists that the perceptron learning visits that state and the *best* vector out of this set should be returned by the algorithm.

The basic idea of perceptron learning is preserved. If a sample is incorrectly classified the weights are updated by adding (or subtracting) the pattern to the current weights (fixed increment). The training samples *must* now be drawn at random, repeating all training pattern often enough. The pocket algorithm keeps a weight vector aside, the pocket weights, that is the *best* weight vector found so far. If a training pattern is classified correctly by the current weights, the total number of consecutive correct classifications of this vector is compared with that number for the pocket weights. If this value is larger, then the current weights are compared on their performance on the total training set. The pocket weights are replaced by the current weights if the new weights perform better.

The start vector for the pocket vector and the current weight vector can be chosen at random or fixed ($\mathbf{w}=\mathbf{0}$) just like the standard perceptron training. The pocket convergence theorem states that a fixed number of trials are needed to ensure that the pocket weights are the desired optimal set of weights.

The pocket convergence theorem [Gallant 1990]. Let D be a finite set of design patterns $\mathbf{y}$, for which the individual components are restricted to the values $\{-1, 0, 1\}$ and the corresponding class labels (targets) are $\{-1, 1\}$. For a given probability p, a finite number of updates of the pocket algorithm is sufficient to guarantee that with a probability exceeding p, the optimal weights will be found in the pocket vector.

The proof of this theorem is claimed in the appendix of [Gallant 1990] and is omitted here. The convergence theorem has two limitations. The first obvious limitation, that the pattern features are restricted to the values $\{-1, 0, 1\}$, is needed for the proof to ensure that the perceptron procedure is a finite state process. The second severe limitation is that it does not give any hint when to stop the algorithm. The theorem only states that a finite number of updates is needed, but not how many. In practice a *large* number of iteration will be taken, probably a function of the computer resources.

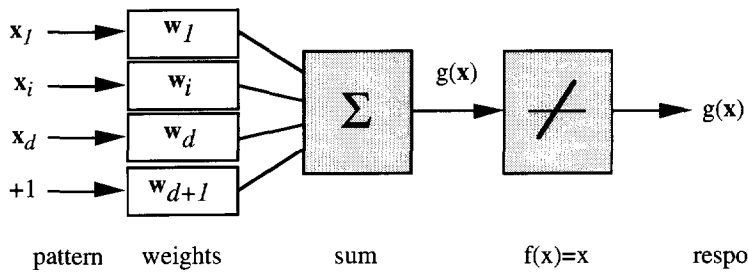

Figure 3·7: The linear perceptron.

Besides the limitation of the convergence proof, this approach is intuitively attractive. The pocket weights are only replaced by the current weights if these are *really* better and therefore performance *ratchets up* during training. In an overlapping situation and with continuous valued inputs the perceptron training may not be described by a finite state machine, but of those vectors that are found the best one is returned. The expense of this procedure is that the number of consecutive correct classifications must be maintained. If this exceeds the number of the pocket weights, the complete training set must be evaluated.

3·4 Linear perceptrons

The perceptrons studied so far all contained a non-differentiable element, a threshold device (figure 3·3) or a maximum selector (figure 3·1). These non-differentiability's impose difficulties for the optimization. Furthermore the criterion function was a function of the misclassified samples only, in order to solve this problem. In the following a criterion function is introduced that is a function of all the samples. The difficulties imposed by the non-linearity are now avoided by directly formulating the criterion as a function of the discriminant function 3·6. An alternative viewpoint is that the threshold element in figure 3·3 is replaced by a linear function $f(x) = x$, leading to the architecture of the linear perceptron shown in figure 3·7.

3·4·1 The minimum squared error procedure

In the previous algorithms the patterns of one class should have a positive response to the discriminant function and the samples of the alternative class should have a negative response. In the following criterion function the samples should approximate a desired target response, a fixed value t for all samples of the same class. A design set is thus a set of training patterns, with the corresponding target responses for every sample. If t_y is the desired response for a pattern $\mathbf{y}$ the following criterion function is proposed:

$$J_{sse}(\mathbf{w}) = \sum_{\mathbf{y}\in D}(t_y - \mathbf{w}\cdot\mathbf{y})^2 = \sum_{\mathbf{y}\in D}(t_y - \mathbf{w}^T\mathbf{y})^2 \tag{3·26}$$

The sum of squared errors is probably one of the most widely used criterion functions, specially outside the field of pattern recognition. The problem of finding the solution that minimizes 3·26 is a classical one and it can be solved explicitly or by gradient search. The gradient search is the straightforward application of equation 3·15 to equation 3·26. This results in the following update rule:

$$\mathbf{w}' = \mathbf{w} - \eta\nabla J_{sse}(\mathbf{w}) = \mathbf{w} + \eta\sum_{\mathbf{y}\in D}(t_y - \mathbf{w}\cdot\mathbf{y})\mathbf{y} \tag{3·27}$$

A stochastic approximation of 3·27 is also possible and is known as the *Widrow-Hoff Delta rule, LMS rule* or *adaline rule* [Widrow 1960]. This is again the sample update version of the steepest descent step:

$$\mathbf{w}' = \mathbf{w} + \eta\ (t_y - \mathbf{w}\cdot\mathbf{y})\mathbf{y} \tag{3·28}$$

The learning rate η for both equations 3·27 and 3·28 is more critical than with the error correction procedures. In section 3·4·3 the stability and convergence of 3·27 will be investigated and bounds for stable convergence will be shown. The problem of finding a proper η can be avoided by solving the least squares formulation explicitly. Equation 3·26 is rewritten as follows:

$$J_{sse}(\mathbf{w}) = \sum_{\mathbf{y}\in D}(t_y^2 + \mathbf{w}^T\mathbf{y}\mathbf{y}^T\mathbf{w} - 2t_y\mathbf{y}^T\mathbf{w}) = \sum_{\mathbf{y}\in D}t_y^2 + \mathbf{w}^T(\sum_{\mathbf{y}\in D}\mathbf{y}\mathbf{y}^T)\mathbf{w} - 2(\sum_{\mathbf{y}\in D}t_y\mathbf{y}^T)\mathbf{w} \tag{3·29}$$

The sum of $\mathbf{y}\mathbf{y}^T$ is, besides a scaling, the estimated *input correlation* matrix and the last summation in 3·29 is similarly, a vector that represents the *input-target correlation.* A solution vector of 3·29 must be a stationary point, so the optimal weight vector $\mathbf{w}^*$ is found where the gradient is zero.

$$\nabla J_{sse}(\mathbf{w}^*) = 0 \ \Rightarrow\ 2(\sum_{\mathbf{y}\in D}\mathbf{y}\mathbf{y}^T)\mathbf{w}^* - 2(\sum_{\mathbf{y}\in D}t_y\mathbf{y}^T) = \mathbf{0} \tag{3·30}$$

If the input correlation matrix is non-singular, the optimal weight vector $\mathbf{w}^*$ called the Wiener solution, can be found by matrix inversion:

$$\mathbf{w}^* = (\sum_{\mathbf{y}\in D}\mathbf{y}\mathbf{y}^T)^{-1}(\sum_{\mathbf{y}\in D}t_y\mathbf{y}^T) \tag{3·31}$$

When the input correlation is singular, a minimum squared error solution still exists, but is not unique. An extensive literature exists on this topic. Methods such as singular value decomposition exist for solving 3·30 when matrix $\mathbf{y}\mathbf{y}^T$ is rank deficient. A full treatment of this topic can be found in chapter 5 of [Golub 1989].

3·4·2 Properties of error space

The sum of squared errors is a quadratic form with a minimum at $\mathbf{w}^*$. In [Widrow 1985] it is shown that 3·29 can also be expressed as:

$$J_{sse}(\mathbf{w}) = J_{min} + (\mathbf{w} - \mathbf{w}^*)^T (\sum_{\mathbf{y} \in D} \mathbf{y}\mathbf{y}^T)(\mathbf{w} - \mathbf{w}^*) \tag{3·32}$$

A quadratic form can always be transformed into a diagonal form, given by the law of inertia (see [Lipschutz 1991, theorem 4·19]). If the input correlation is full rank, a rotation of 3·32 onto the eigenvectors of the correlation matrix, will result in a diagonal expression for the error criterion.

Because the input correlation matrix is real and symmetrical the eigenvectors are orthogonal (see [Lipschutz 1991, theorem 8·13]). When equation 3·32 is brought into its diagonal form and second order derivatives are calculated, it is easily shown that this Hessian matrix is twice the matrix with the eigenvalues ([Widrow 1985, chapter 3]) as diagonal terms (equation 3·39 multiplied by two for the general case).

The error space, the geometrical form of the $J_{sse}()$ criterion as function of the weight vector, is well behaved. The error surface is convex with one minimum, furthermore the orientation and form are completely determined by the eigenvectors and eigenvalues of the $\mathbf{y}\mathbf{y}^T$ matrix. This is a favorable feature for a gradient descent, It ensures that the global minimum is reachable; there are no local minima.

3·4·3 Stability and convergence properties

The stability and convergence properties will first be examined for a one dimensional weight space. The squared error criterion of 3·32 reduces to the following scalar equivalent:

$$J_{sse}(w) = J_{min} + (w - w^*)\lambda(w - w^*) \tag{3·33}$$

The steepest descent rule of 3·15 applied to this form, results in the following expression:

$$w' = w - 2\eta\lambda(w - w^*) = (1 - 2\eta\lambda)w + 2\eta\lambda w^* \tag{3·34}$$

This equation is a linear, ordinary difference equation and the solution or stability requirements follow from linear system theory ([Oppenheim 1983, page 655]). It can also be solved by induction from a few iterations, starting from the initial guess $w^{(0)}$. The generalization to the i-th iteration is ([Widrow 1985, page 49]):

$$w^{(i)} = w^* + (1 - 2\eta\lambda)^i (w^{(0)} - w^*) \tag{3·35}$$

The convergence properties are determined by $(1\text{-}2\eta\lambda)$, called the *geometrical ratio.* If η is chosen as:

$$|1-2\eta\lambda| < 1 \Rightarrow 0 < \eta < \frac{1}{\lambda} \tag{3·37}$$

the difference equation 3·35 is clearly stable and converges to the optimal solution w^*. Besides a stable learning, the geometrical ratio also determines the learning speed. If η is chosen as $\lambda/2$, the optimal solution is reached in a single step. If η is chosen small then a considerable number of training steps must be taken to reach the optimal solution.

The vector equivalent can be similarly obtained by starting with equation 3·32. In [Widrow 1985, page 60] the following expression is derived:

$$\mathbf{w}^{(i)} = \mathbf{w}^* + (\mathbf{1} - 2\eta \sum_{\mathbf{y}\in D} \mathbf{y}\mathbf{y}^T)^i (\mathbf{w} - \mathbf{w}^*) \tag{3·38}$$

In the previous section it was stated that the input correlation matrix can be written in a diagonal form using the eigenvectors as a new basis. Assume that $\mathbf{Q}$ is the square matrix with columns that are the eigenvectors and Λ is a diagonal matrix with the corresponding eigenvalues on the diagonal. The input correlation decomposition is then given by:

$$\sum_{\mathbf{y}\in D} \mathbf{y}\mathbf{y}^T = \mathbf{Q}\Lambda\mathbf{Q}^{-1} \tag{3·39}$$

With some matrix algebra and using $\mathbf{Q}\mathbf{Q}^{-1}=\mathbf{I}$, equation 3·38 can be rewritten in the form [Widrow 1985, page 60]:

$$\mathbf{w}^{(i)} = \mathbf{w}^* + \mathbf{Q}(\mathbf{1} - 2\eta\Lambda)^i \mathbf{Q}^{-1}(\mathbf{w} - \mathbf{w}^*) \tag{3·40}$$

The convergence properties are now determined by $(\mathbf{1} - 2\eta\Lambda)$ which is a diagonal matrix. Since the product of two diagonal matrices is a diagonal matrix with products of the corresponding elements, a set of equations 3·37 can converge. If λ_{max} is the largest eigenvalue of the input correlation matrix, stable convergence requires an η such that:

$$|1 - 2\eta\lambda_{max}| < 1 \Rightarrow 0 < \eta < \frac{1}{\lambda_{max}} \tag{3·41}$$

The speed of convergence is now determined by the smallest eigenvalue and determines the number of training sweeps. Because the largest eigenvalue limits the choice of the learning rate and the smallest eigenvalue limits the speed of convergence, the ratio of these two quantifies the system. This ratio, $\lambda_{max}/\lambda_{min}$ is often called the *condition number.*

As stated earlier in section 3·4·2, the second derivatives in the direction of the eigenvectors of the error space are equal to twice the eigenvalues. The eigenvalues determine the convergence properties and the range for η, for which the training is

stable. Therefore the second order properties determine the steepest descent learning characteristics.

3·4·4 Asymptotic properties of linear perceptron

The minimum squared error procedure has good convergence properties, one minimum and a convex error space. If the learning rate is chosen properly, the algorithm will approximate the optimal solution. The optimality of the solution is in a squared error sense, which is different from the minimum error classifier (section 2·3·3). The relation between the Bayes classifier and the minimum squared error classifier will be shown.

In the following the sample estimate of the sum of squared errors, is replaced by the mean squared error function:

$$J_{mse}(\mathbf{w}) = E[(t_y - \mathbf{w}\cdot\mathbf{y})^2] = \int \{\sum_{i=1}^{c} (t_y - \mathbf{w}\cdot\mathbf{y})^2 \, p(\mathbf{y},\omega_i)\} d\mathbf{y} \tag{3·42}$$

Substituting $p(\mathbf{y},\omega_i) = P(\omega_i \mid \mathbf{y})p(\mathbf{y})$ in 3·42 yields:

$$J_{mse}(\mathbf{w}) = \int \{\sum_{i=1}^{c} (t_y - \mathbf{w}\cdot\mathbf{y})^2 P(\omega_i \mid \mathbf{y})\} p(\mathbf{y}) d\mathbf{y} = E[\sum_{i=1}^{c} (t_y - \mathbf{w}\cdot\mathbf{y})^2 P(\omega_i \mid \mathbf{y})] \tag{3·43}$$

When the squared expression in 3·43 is expanded and simplified by exploiting that $\mathbf{w}\cdot\mathbf{y}$ is independent of ω_i the mean squared error can be formulated as:

$$J_{mse}(\mathbf{w}) = E[(\mathbf{w}\cdot\mathbf{y})^2 - 2\mathbf{w}\cdot\mathbf{y}\sum_{i=1}^{c} t_y P(\omega_i \mid y) + \sum_{i=1}^{c} t_y^2 P(\omega_i \mid y)] \tag{3·44}$$

This can be written into the final desired form:

$$J_{mse}(\mathbf{w}) = E[(\mathbf{w}\cdot\mathbf{y} - \sum_{i=1}^{c} t_y P(\omega_i \mid \mathbf{y}))^2] + E[\sum_{i=1}^{c} t_y^2 P(\omega_i \mid \mathbf{y})] - E^2[\sum_{i=1}^{c} t_y P(\omega_i \mid \mathbf{y})] \tag{3·45}$$

The second and third expectations in 3·45 are independent of $\mathbf{w}$, so the optimization of this expression is equal the optimization of the first term. Lets analyze this criterion for a two-class classification problem with the target values as follows:

$$\begin{aligned} \mathbf{y} \in \omega_1 &\rightarrow t_y = 1 \\ \mathbf{y} \in \omega_2 &\rightarrow t_y = -1 \end{aligned} \tag{3·46}$$

The mean squared error procedure thus optimizes the linear perceptron such that the perceptron response approximates, in the mean squared error sense, the following function:

$$\mathbf{w}\cdot\mathbf{y} \approx P(\omega_1 \mid \mathbf{y}) - P(\omega_2 \mid \mathbf{y}) \tag{3·47}$$

The right-hand side of this expression is the minimum error or Bayes classifier for the two-category case. This result gives considerable insight into the linear perceptron,

because it reveals the relation with the statistical pattern recognition approach. There are two important practical remarks to be made from this conclusion.

First we should not forget that the sum of squared errors criterion (3·26) is besides a scaling, an estimate of the mean squared error (3·42). The optimality of the solution thus depends on the sample size.

The second and more serious remark concerns the fact that the *best* approximation in the squared error sense, does not necessarily minimize the probability of error. Formula 3·45 implies that a linear regression is performed with the difference of the *posteriori* probabilities, that is not necessarily a linear function. A regression with a function from a different family than $\mathbf{w} \cdot \mathbf{y}$ can introduce a bias, resulting in a higher classification error. Unfortunately one can not guarantee that this approach will find the Bayes linear classifier for all distributions, only for some particular cases.

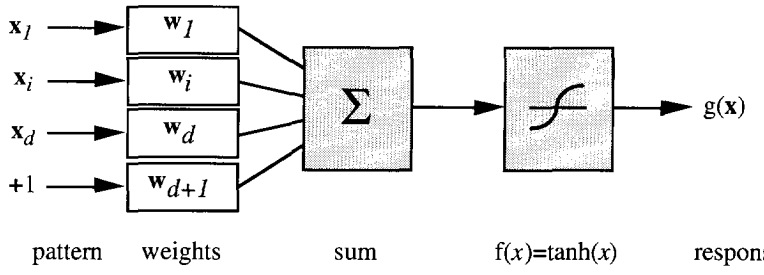

Figure 3·8: The non-linear perceptron

3·5 The non-linear perceptron

It is straightforward to generalize the linear perceptron to a non-linear perceptron, by replacing the linear output function with any general differentiable function (see figure 3·8). The functions that are commonly used are bounded and monotonic increasing functions such as:

$$f(x) = \tanh(x) \tag{3·48}$$

$$f(x) = \frac{1}{1+e^{-x}} \tag{3·49}$$

The mathematical description of the non-linear perceptron is hence:

$$g(\mathbf{y}) = f(\mathbf{w} \cdot \mathbf{y}) \tag{3·50}$$

The capability of the non-linear perceptron, with a monotonic increasing output function, is clearly equal to the TLU and linear perceptron. It therefore does not seem to have any advantage over the linear perceptron, since only linearly separable problems can be solved. The non-linear perceptrons are, however, important in the multi-layer

networks, where they are the building blocks (see chapter 4). In the following sections some of the consequences of the use of a smooth non-linear output function will be investigated.

3·5·1 Gradient descent learning

The training algorithm for the non-linear perceptron is similar to the linear perceptron. The sum of squared errors is again used as a criterion function, as follows:

$$J_{sse}(\mathbf{w}) = \sum_{\mathbf{y}\in D}(t_{\mathbf{y}} - g(\mathbf{y}))^2 = \sum_{\mathbf{y}\in D}(t_{\mathbf{y}} - f(\mathbf{w}\cdot\mathbf{y}))^2 \tag{3·51}$$

The steepest descent rule 3·15 applied to this criterion function results in:

$$\mathbf{w}' = \mathbf{w} - \eta\nabla J_{sse}(\mathbf{w}) = \mathbf{w} + \eta\sum_{\mathbf{y}\in D}(t_{\mathbf{y}} - f(\mathbf{w}\cdot\mathbf{y}))f'(\mathbf{w}\cdot\mathbf{y})\mathbf{y} \tag{3·52}$$

A stochastic approximation of this update-rule is of course also feasible. The sample-update version is given by:

$$\mathbf{w}' = \mathbf{w} + \eta(t_{\mathbf{y}} - f(\mathbf{w}\cdot\mathbf{y}))f'(\mathbf{w}\cdot\mathbf{y})\mathbf{y} \tag{3·53}$$

The learning rate η in both formulas is even more critical (see section 4·3·2) than for the linear perceptron. The error space is *not* well behaved and is mostly not convex. It is possible for the cost function 3·51 to have *local minima* ([Hertz 1991, page 108]). The initial weight vector of the training algorithm and the order of presentation of the design vectors for stochastic training, seems to influence the final solution.

A rigorous proof of the convergence properties of the steepest descent, as there exists for the linear perceptron, does not exist for the non-linear perceptron. The criterion function 3·51 can have local minima and therefore it is possible that the learning is converges to a sub-optimal solution.

The stability conditions of the training algorithm, for the linear perceptron dependent on the eigenvalues of the input correlation, has now become complicated. The *local* behavior of the training system can be analyzed by approximating 3·50 with a first order Taylor series in $\mathbf{w}$ and requiring the this should be a stable system as derived for the linear perceptron. This approach is identical to the absolute stability criterion, as formulated for the numerical integration of a general non-linear differential equation (see [Gear 1971, page 9]).

An explicit solution, such as the Wiener solution of formula 3·31, does not exist either. The steepest descent algorithm or another optimization technique must be used to find a (local) minimum of 3·51.

3·5·2 Properties of error space

The criterion functions found so far were functions linear (3·16), linear to the decision surface normal (3·19) or quadratic (3·26) with respect to the misclassified pattern $\mathbf{y}$. If we want to obtain a minimum error or minimum risk classifier, it would be

better to choose the criterion function accordingly. For a minimum error classifier the following criterion function is preferred:

$$J_{err}(\mathbf{w}) = \sum_{\{\mathbf{y} \in D | \alpha(\mathbf{y}) \neq \omega_i\}} 1 \tag{3·54}$$

In section 3·4·4 it was argued that the linear perceptron trained with the sum of squared errors criterion (3·26), does not always result in the classifier with the lowest probability of error. We can formulate this more explicitly as: the minimum value of **w** for criterion 3·26 does not coincide with that for 3·54.

The error that is associated with a (misclassified) sample is an important issue. The contribution of a *single* sample to the error function is sometimes called the *pattern error function*. Three pattern error functions 3·16, 3·26 and 3·51 are shown in figure 3·9. For 3·51 the non-linear function 3·49 was used. These curves represent a trainable system with one adjustable parameter, $w=2$ and a target value $t=0$.

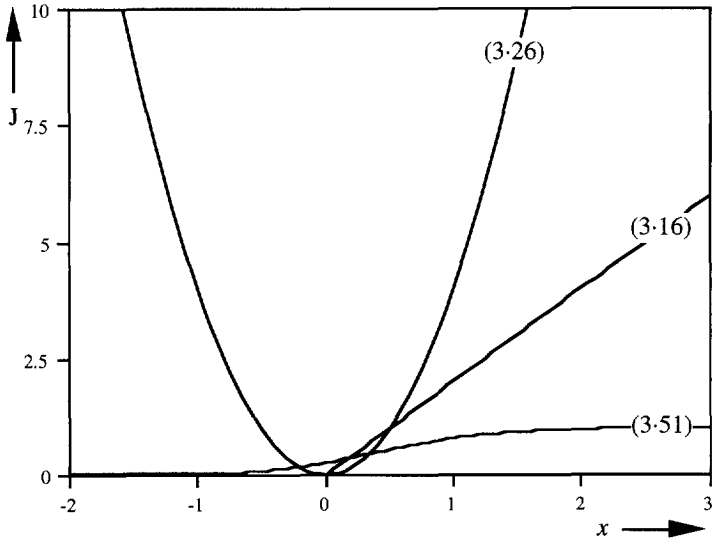

Figure 3·9: Pattern error functions 3·16, 3·26 and 3·51 plotted as function of a one dimensional feature x and for a target value 0. The trainable pattern classifier is a system with one parameter, set to w=2.

The pattern error function for the minimum error criterion is a unit step function, as shown in figure 3·10. The other pattern error functions are dependent on the weight vector, in particular the norm of this vector. The error function of 3·51 for the sigmoid function 3·49 is shown in figure 3·10, for different values of the weight vector. The increasing weight value w is equivalent to an increasing norm of the weight vector **w** for the non-linear perceptron. In this figure it is shown that if the norm of the weight vector increases, the pattern error function for the non-linear perceptron approaches that of the minimum error function.

This is an important observation, noticed for example by [Raudys 1992]. The linear perceptron, together with the sum of squared errors criterion, can be biased towards a classifier with a higher error rate. A squashing function like 3·48 or 3·49 can reduce this bias and if the training algorithm returns a vector with a relatively large norm, this bias might be negligible. In other words, the global minimum of this sum of squared errors coincides with the minimum error criterion.

This property is important because in pattern recognition the minimum error or risk is what we want to minimize. However, the error for the complete design set can be non-convex and there is no guarantee that the training algorithm will find this solution. The conclusion is that the bias of the classifier can be removed with the risk that the training stops in a sub optimal state.

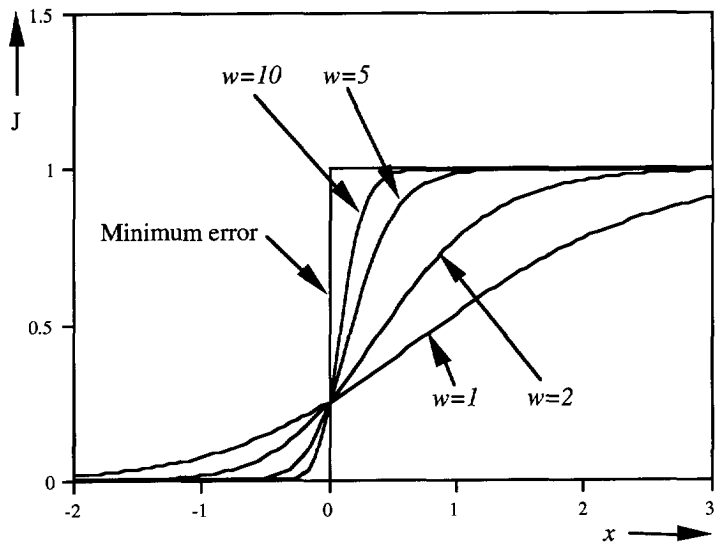

Figure 3·10: The pattern error function for a non-linear perceptron with squashing function 3·49 for different weight values. The minimum error function is plotted for comparison.

3·5·3 A training example

An interesting example is an artificially generated pattern data set proposed by [Highleyman 1962]. This set is a two-dimensional Gaussian distributed data set, with two equal probable classes and the following parameters for the pattern distribution:

$$\omega_1 : \ \bar{\mathbf{x}}_{\omega_1} = (1,\ 1)^T,\ \mathbf{S}_{\omega_1} = \begin{pmatrix} 1 & 0 \\ 0 & 0.25 \end{pmatrix}$$
$$\omega_2 : \ \bar{\mathbf{x}}_{\omega_2} = (2,\ 0)^T,\ \mathbf{S}_{\omega_2} = \begin{pmatrix} 0.01 & 0 \\ 0 & 4 \end{pmatrix} \qquad (3\cdot55)$$

An example training set is plotted in figure 3·11, showing 100 samples for each class. The minimum error classifier or Bayes classifier has an error of $\varepsilon^* \approx 5.5\%$ and is achieved with a quadratic classifier of the type 2·24. In [Wolff 1966] this data set is investigated in depth. He found that the best linear classifier with error $\varepsilon \approx 11\%$, is given by (see figure 3·11):

$$g(\mathbf{x}) = -100\mathbf{x}_1 + 1.12\mathbf{x}_2 + 176.7 \tag{3·56}$$

Furthermore [Wolff 1966] found that a sub optimal or local minimum solution is often found, with an associated error $\varepsilon \approx 22\%$, given by the following discriminant function (see also figure 3·11):

$$g(\mathbf{x}) = -104\mathbf{x}_1 + 100\mathbf{x}_2 + 132 \tag{3·57}$$

To visualize the error criterion for the linear and non-linear perceptron, figure 3·12 is constructed as follows. The optimal linear classifier given by 3·56 is rotated around an arbitrary chosen point $(1.76, 0)^T$ and the criterion function for both perceptrons is measured on a training set with 100 sample for each class. This value is measured for different rotation angles, together with the true probability of error of the corresponding classifier.

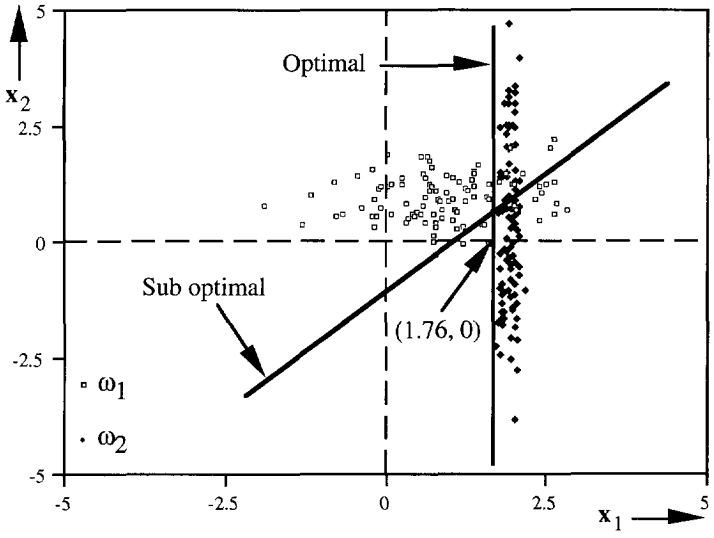

Figure 3·11: An example training set of 200 samples for the Highleyman (3·55) distribution. The optimal (3·56) and the sub-optimal (3·57) perceptron solutions are shown.

From figure 3·12 some interesting properties are observed. The error curve for the linear perceptron is convex and smooth, as predicted in section 3·4·2, but the minimum value does not correspond to the minimum error classifier. The Wiener solution $\mathbf{w}^*$, the

perceptron corresponding to the minimum of this curve, is approximately equal to the classifier given by 3·57 and has a corresponding error of $\varepsilon \approx 24\%$.

For the non-linear perceptron the minimum coincides with the minimum error classifier and the forms of these error functions show large similarity. At the right side of the minimum an almost flat region is found. This region corresponds to the sub optimal solution 3·57, found by [Wolff 1966]. This local minimum is a consequence of the true distributions and is not introduced by the chosen criterion. Training a non-linear perceptron with the steepest descent rule 3·52 often stops in this region. The stochastic method 3·53 seems to be more robust, because figure 3·12 is only the expected value now.

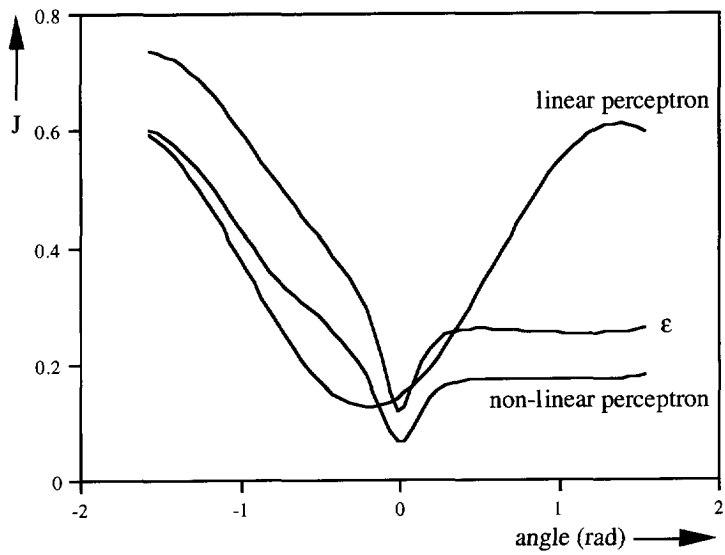

Figure 3·12: Three criterion functions plotted as function of the optimal linear classifier, rotated with a given angle around point (1.76, 0) in pattern space. The criterion functions are the probability of error and the sum of squared errors for the linear and non-linear perceptron. The values of the linear perceptron are scaled to fit into the graph.

In figure 3·10 the pattern error function was shown for different weight values, corresponding to different norms of the weight vector **w** for the multi-dimensional case. In figure 3·13 this is shown for the non-linear curve of figure 3·12.

In figure 3·13 it can be seen that for small weights the criterion function is convex because the non-linear perceptron is approximately a linear perceptron. The minimum is biased, but shifts to the minimum error classifier when the norm is increased. When the norm is increased again the criterion function becomes more equal in shape with the minimum error criterion. This agrees with the conclusion from the previous section. If the norm is increased again, the curve becomes rough, showing the individual

contributions to the error of each pattern, a result of the finite design set (200 samples). Some small local minima now become visible.

This phenomenon shows similarities with the Parzen window approach to estimate probability densities (see [Duda 1973, page 88]). In this approach a *window width* or *smoothness parameter* influences the estimate. A small value will result in sharp pulses and a large window gives an estimate with little resolution. Two examples are shown in [Duda 1973, page 93-94].

The criterion function shown in figure 3·13 can not become so erratic as with Parzen windows because the pattern error function approaches a step function and not a Dirac delta function. The derivative of this figure, however, does show a superposition of sharp pulses, dependent on the position of the samples.

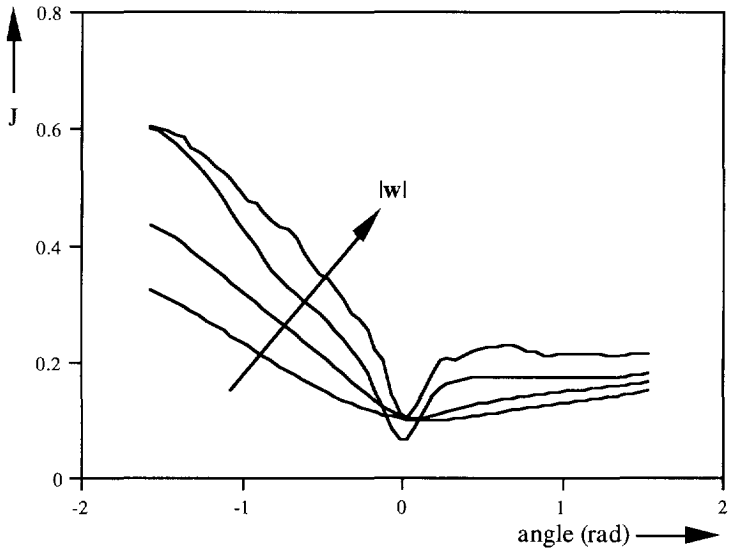

Figure 3·13: The criterion function of the non-linear perceptron, similar as show in figure 3·12, but for increasing norm |w| *of the weight vector.*

3·6 Generalization capabilities

The expected probability of error of a classifier is probably a more important property than asymptotic behavior or convergence properties. In practical situations one needs to know how large the training set should be and what performance can be expected when such a system is used. In this section the relevant theory and tables for the pattern classifiers discussed so far are presented.

3·6·1 Capacity of perceptrons

A classifier is trained with only m samples and the classification is based on the knowledge extracted from these examples. In operation a pattern classifier will receive patterns never seen before and a decision is based on the generalization or extrapolation from the training data. For a given situation one classifier will generalize better than another. The importance of having an over-determined system, m is large with respect to the type of classifier, is obvious.

The question, however, is how to quantify this into numbers. A classical result is derived by [Cover 1965], already mentioned in section 2·4·6. Suppose that m samples are in a *general position* in a d dimensional space, meaning that no subset of $d+1$ points falls in a $d-1$ dimensional subspace. Each point is labeled either ω_1 or ω_2. Of the 2^m possible combinations, a certain fraction will permit a linear dichotomy. These are the labelings for which a linear hyperplane exists that separates these points with zero error. In [Cover 1965] it is shown that this fraction is given by:

$$P_{lin}(m,d) = \begin{cases} 1 & m \le d+1 \\ \dfrac{2}{2^m}\displaystyle\sum_{i=0}^{d}\binom{m-1}{i} & m > d+1 \end{cases} \qquad (3\text{·}58)$$

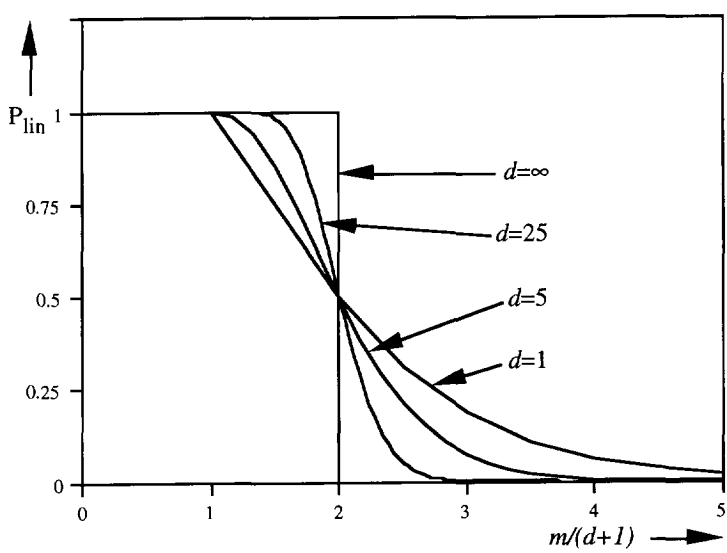

Figure 3·14: The fraction of linear dichotomies of m points in d dimensions.

Formula 3·58 is shown in figure 3·14 as a function of the fraction $m/(d+1)$. All graphs pass through $m=2(d+1)$, which suggests defining this as the *capacity* of a perceptron. The probability that a random labeling of $m=2(d+1)$ points is a linear dichotomy is 0.5, so it can be concluded that this is still not a very over-determined

situation. This observation leads to the guideline that the number of training patterns should be chosen larger than this critical point.

The sample size consideration formulated by [Cover 1965] gives insight into the information represented by m samples for the linear classifier. It does not give any information about the expected probability of error of a classifier when trained with such a number of samples. In [Foley 1972, page 615] simulation results are reported, that show the resubstitution error $\hat{\varepsilon}_r$ as function of the fraction $m/(d+1)$ for a particular distribution of patterns that are labeled at random. If in practice an observed resubstitution error is close to the value found by Foley, one must be suspicious about the performance of the classifier for the complete population.

3·6·2 Sample size and probability of error

The expected probability of error ($E[\varepsilon_m]$) is dependent, as discussed in section 2·4·4, on the chosen criterion function J() and the method used to optimize the perceptron weights. For Gaussian distributed patterns, with common covariance matrix but with different means, asymptotic expressions and tables are available. In the following it is assumed that both classes are equally probable and $m/2$ samples per class are available. For this distribution the linear classifier is the best possible classifier so for very large sample sizes this should approach the Bayes error ε^*.

In [Smith 1972] asymptotic expressions are derived for three *stochastic* gradient descent algorithms. These expressions have the following common form:

$$E[\varepsilon_m] = \varepsilon^* + \frac{a_0 + a_1(d-1)}{m} \tag{3·59}$$

In this formula ε^* is the Bayes error of the asymptotic optimal classifier, m is the total number of training samples and d is the pattern dimensionality. The parameters a_0 and a_1 are dependent on the Mahalanobis distance (δ) and the optimized criterion. The coefficients for the criterion functions discussed in this thesis, equations 3·16 and 3·26, are given by table 3·1.

Table 3·1: The coefficients for equation (3·59).

		J_{sse}	(Eq. 3·26)	J_{fi}	(Eq. 3·16)
δ	ε^*	a_0	a_1	a_0	a_1
1.0	0.309	0.0880	0.4400	0.0152	0.505
2.0	0.159	0.1210	0.2420	0.1960	0.317
4.0	0.023	0.0540	0.0675	0.2180	0.232

For the linear perceptron, trained with the matrix inverse technique of equation 3·31, the simple expansion from Raudys 2·21 is valid. In table 3·2 the ratio $E[\varepsilon_m]/\varepsilon^*$ is tabulated as function of sample size, pattern dimension and Mahalanobis distance. The values found in this table are taken from [Raudys 1980].

Table 3·2: The ratio $E[\varepsilon_m]/\varepsilon^*$ as function of d, m and δ for the matrix inverse solution of the J_{sse} criterion 3·26 ([Raudys 1980]).

d=3	δ					*d=5*	δ				
	1.68	2.56	3.76	4.65	5.50		1.68	2.56	3.76	4.65	5.50
m						*m*					
4	2.00	3.26	8.27	20.6	59.3	6	2.11	3.64	9.90	25.7	76.1
6	1.64	2.22	4.17	8.00	17.9	8	1.80	2.65	5.62	11.9	28.9
12	1.31	1.50	2.00	2.74	4.07	10	1.64	2.21	4.01	7.37	15.4
30	1.12	1.17	1.30	1.45	1.66	20	1.32	1.51	2.00	2.66	3.78
60	1.06	1.08	1.14	1.20	1.27	50	1.13	1.18	1.31	1.45	1.64
120	1.03	1.04	1.07	1.09	1.13	100	1.06	1.09	1.14	1.20	1.27
300	1.01	1.02	1.03	1.04	1.05	200	1.03	1.04	1.07	1.10	1.13
						500	1.01	1.02	1.03	1.04	1.05

d=8	δ					*d=12*	δ				
	1.68	2.56	3.76	4.65	5.50		1.68	2.56	3.76	4.65	5.50
m						*m*					
10	2.04	3.38	8.57	20.9	58.2	14	2.12	3.64	9.71	24.6	70.6
12	1.85	2.78	6.00	18.8	30.9	20	1.75	2.50	4.88	9.41	20.2
16	1.63	2.19	3.87	6.84	13.5	24	1.63	2.18	3.77	6.51	12.4
32	1.32	1.51	1.97	2.59	3.58	48	1.33	1.51	1.96	2.54	3.46
80	1.13	1.18	1.31	1.45	1.63	120	1.13	1.19	1.31	1.45	1.62
160	1.07	1.09	1.14	1.20	1.27	240	1.07	1.09	1.14	1.20	1.27
320	1.03	1.04	1.07	1.10	1.13	480	1.03	1.04	1.07	1.10	1.13
800	1.01	1.02	1.03	1.04	1.05	1200	1.01	1.02	1.03	1.04	1.05

d=20	δ					*d=50*	δ				
	1.68	2.56	3.76	4.65	5.50		1.68	2.56	3.76	4.65	5.50
m						*m*					
24	2.06	3.44	8.73	21.1	57.4						
32	1.78	2.56	5.06	9.76	20.9	60	2.05	3.39	8.40	19.7	52.0
40	1.63	2.16	3.69	6.22	11.4	100	1.62	2.15	3.61	5.95	10.6
80	1.33	1.51	1.95	2.50	3.36	200	1.33	1.51	1.93	2.47	3.27
200	1.14	1.19	1.31	1.44	1.62	500	1.14	1.19	1.31	1.44	1.61
400	1.07	1.09	1.15	1.20	1.27	1000	1.07	1.09	1.15	1.20	1.27
800	1.03	1.05	1.07	1.10	1.13	2000	1.04	1.05	1.07	1.10	1.13
2000	1.01	1.02	1.03	1.04	1.05	5000	1.01	1.02	1.03	1.04	1.05

The approximately *1/m* proportionality predicted by 3·59, can for small sample sizes be found in these tables in the columns corresponding to one Mahalanobis distance and the same pattern dimensionality.

The values that are tabulated (and formula 3·59) correspond to the ideal Gaussian case. If the data is not spherically normal ($\mathbf{S} \propto \mathbf{I}$) then significantly greater or smaller values can be measured. It is the hope [Raudys 1980, page 244] that in practical situations this is rare and that these tables and approximation formulas are good guidelines for the pattern recognition practitioner.

3·7 Beyond simple perceptrons

3·7·1 Introduction

From a mathematical point of view the perceptrons or TLU's found so far, have limited capabilities. The pattern recognition task needs to be perfectly linearly separable or the minimum error classifier should be linear. The perceptrons are only optimal for these special cases, a consequence of the simple structure of these classifiers.

After the publications of [Rosenblatt 1962], perceptrons have been widely described as pattern recognition or learning machines and as such have been discussed in a large number of books, articles and reports. Most of these writing neglected the limitations but focussed on the adaptation or learning capabilities. In the famous book by Minsky and Papert [Minsky 1969] this was noticed and the excitement and exaggerated claims of this field of research were heavily criticized.

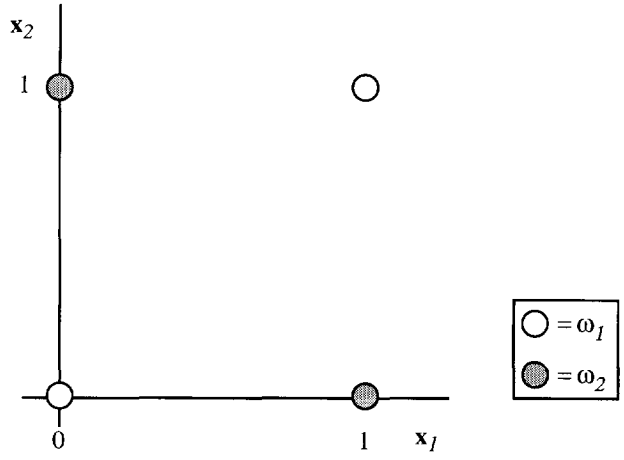

Figure 3·15 The exclusive-or problem.

The learning capabilities induced great enthusiasm, specially from those who were attracted by the flexibility of biological systems that learn from their environment. In their book [Minsky 1969] showed that some simple biologically inspired problems could not be solved by a perceptron with limited connections. The conclusion of

Minsky and Papert from this simple example was, that they could not see any value from experiments that paid no attention to the limitations [Minsky 1969, page 18]. Their publication nearly eliminated the entire line of research in this field.

From a pattern recognition point of view this discussion can be formulated in a specific classification problem known in engineering as the parity problem. Assume a pattern vector $\mathbf{x}$ with binary features $\{0, 1\}$. The parity problem states that pattern $\mathbf{x}$ is element of class ω_1 if it contains an even number of 1's and is element of ω_2 otherwise. For a two dimensional vector this problem is also known as the exclusive-or problem.

In figure 3·15 the exclusive-or classification is shown. This figure clearly illustrates that this problem can not be solved with a linear classifier. If Cover's formula 3·58 is applied to this configuration, a fraction 14/16 of the labelings of 4 points in a two dimensional space is linearly separable. The dichotomy shown in figure 3·15 and the inverse labeling, are precisely those two combinations that are not linearly separable. It is disappointing that such a simple example can not be solved by the systems discussed so far.

It is questionable if the d dimensional parity problem is an important pattern recognition example in practice. In pattern recognition often a certain compactness is assumed: patterns found close in *distance* belong to the same class. A large number of methods assume compactness and some examples are cluster analysis [Jain 1988a], nearest neighbor techniques [Duda 1973, page 95] and the famous Fisher linear discriminant [Duda 1973, page 114]. The parity problem is an extreme distribution, all the samples near in (Hamming) distance, belong to another class.

The parity problem is probably not an important pattern recognition problem. It is, however, realistic that a given data set is not linearly separable and that the optimal decision boundary is a non-linear function. Before the publication of Minsky and Papert this was already realized and a number of techniques were proposed to increase the complexity of the perceptron. In the following sections a few solutions are given.

3·7·2 Quadratic machines and functional machines

The quadratic classifier of equation 2·24 can be written into the following form:

$$g(\mathbf{x}) = \sum_{i=1}^{d} \mathbf{W}_{i,i}\mathbf{x}_i^2 + \sum_{i=1}^{d-1}\sum_{j=i+1}^{d} \mathbf{W}_{i,j}\mathbf{x}_i\mathbf{x}_j + \sum_{i=1}^{d} \mathbf{w}_i\mathbf{x}_i + \mathbf{w}_{d+1} \qquad (3\cdot60)$$

This discriminant can be implemented as a perceptron if the quadratic and cross terms of this last equation are presented to the system as *new* features. This is similar to how the augmented training pattern $\mathbf{y}$ is used to train the threshold weight. The quadratic perceptron is shown in figure 3·16 and the quadratic feature vector is calculated here by a special processor. The quadratic perceptron has $(d+1)(d+2)/2$ parameters, a number that becomes impractical for large values of d.

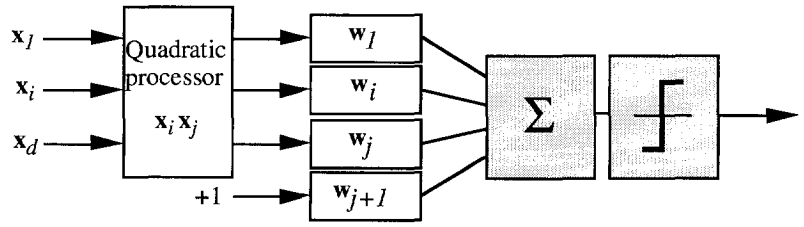

Figure 3·16: The quadratic perceptron.

The idea to couple a special processor before the TLU or perceptron leads to the following general machine. The quadratic processor can be replaced by any processor that implements a one-to-one transformation of the original d dimensional feature vector to a new d' dimensional vector. The dimensionality d' can be larger or smaller than the original pattern dimension d.

The *functional machine* or *Φ machine* can be formulated as:

$$g(\mathbf{x}) = \sum_i \mathbf{w}_i f_i(\mathbf{x}) \tag{3·61}$$

The functions $f_i()$ in discriminant 3·61 can be chosen arbitrarily as long as they do not contain parameters that need to be optimized. The training theorem of section 3·3·5 and the capacity discussion of section 3·6·1 can obviously be extended for this architecture (see [Nilsson 1965, page 38]).

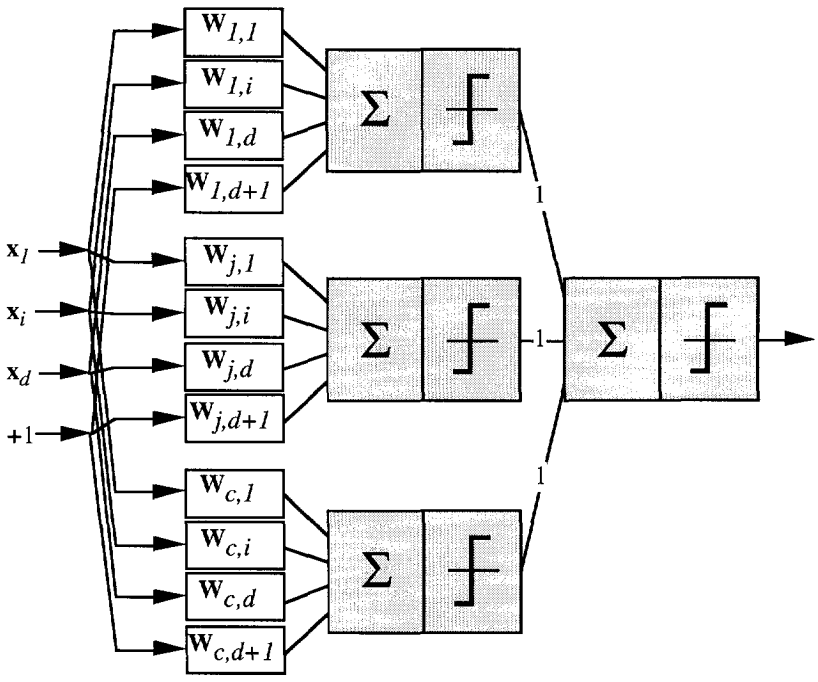

Figure 3·17: A committee machine with three units in the first layer.

At first glance the functional machine seems to solve the limited capabilities of the TLU's and perceptrons. The problem is, however, how to choose the functions $f_i()$ in formula 3·61. A proper choice of these functions requires the full knowledge of true distributions and the non-parametric perceptron approach becomes a parametric pattern recognition method.

3·7·3 Layered machines

There are two fundamental approaches to increase the TLU or perceptron capabilities. One approach, discussed in the previous section, increases the input space with new, non-linear functions of the original features. These special inputs are sometimes called *functional links.*

In the alternative approach the perceptrons are used as building blocks, shortly called units, to create a network of TLU's. A special class of TLU networks is the layered machine, where the TLU's are organized in layers. Each unit in the first layer receives as inputs the pattern to be classified. The outputs of the first layer are used as inputs to the second layer of TLU's. The TLU's of the second and subsequent layers have as their inputs the outputs of the previous layer. The output of the unit in the last layer is the response of the machine.

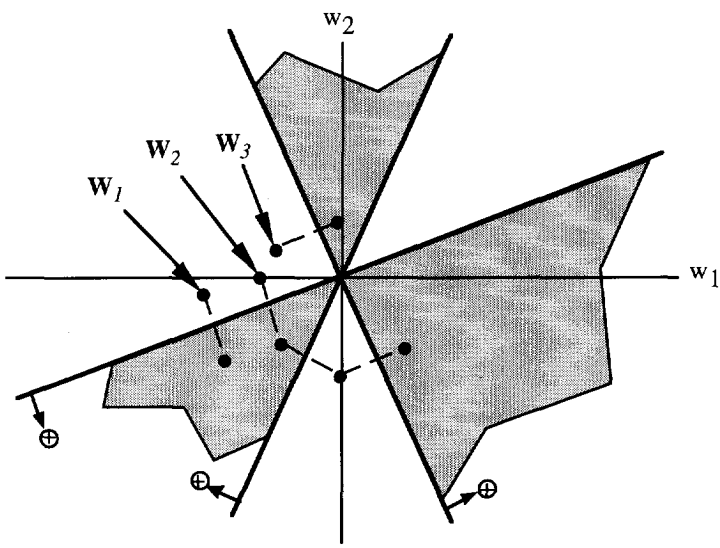

Figure 3·18: The committee training procedure shown for three pattern hyper planes, in a two dimensional two dimensional weight space.

The general idea is to train a layered machine by adjusting all the weights. An efficient training rule was, however, not known before 1985. For a special configuration an iterative training procedure existed but this does not exploit the full capabilities of

the layered machine. This special configuration is shown in figure 3·17 and is known as the *committee machine*.

The committee machine (figure 3·17) consists of two layers. One layer is connected to the features and in a second layer only one unit determines the response. The weights of the output unit are fixed to the value *1*. The output of the TLU's is restricted to *+1* and *-1*. With any odd number of units in the first layer, the machine output is *+1* if the majority of units in the first layer is *+1* and vice versa. The special fixed connection of the output unit therefore implements a majority vote device.

The training procedure adjusts the weight vectors until the design set is successfully classified. The weight vectors for the machine shown in figure 3·17 are initially set to random values. The patterns are again trained one at the time and a procedure similar to the error correction method is used.

No adjustment of the weights is performed if a pattern is classified correctly. If a sample is misclassified than the minimum amount of TLU's that must reverse in response, to achieve a correct classification, is calculated. For the machine shown in figure 3·17 this is either one or two units and only this number of units is updated. The TLU's that will be adjusted are those from the set of incorrect responding units, that have dot products $\mathbf{W}_i \cdot \mathbf{y}$ that are closest to zero. The update rule of 3·10 is applied to those units and the learning rate can be one of the three alternatives discussed in section 3·3·2.

An interpretation of the training algorithm in weight space, similar to the interpretation of section 3·3·4, can be formulated for the training of a committee machine. In figure 3·18 the weight space is shown with three pattern hyper planes for the committee machine. A solution region is not only that region for which all units have a positive response, now all regions for which the majority of units have a positive response. This is shown in figure 3·18 where the three shaded regions are solution regions.

It is clear from this figure that a committee machine is an architecture with more capabilities for separating a data set, because there is more freedom in weight space. The decision regions are in general not convex [Nilsson 1965, page 96]; an improvement over the systems seen so far.

Training now consists of moving all three weight vectors for the machine with three units in the first layer to one of these solution regions. A weight update will move a unit towards one of the solution regions (3·24), but a collective convergence can *not* be guaranteed. The convergence theorem of section 3·3·5 requires that the training sequence occurs often enough, that every unit is continuously trained until the weight vector reaches a solution region. A unit is only updated if it is closest to zero and it is possible that this will never happen. This can occur if the weight vector of one unit is very large compared to the other units.

A result of this last conclusion is that the *initial* weights should be chosen carefully. It is thereby possible that the training will not find a solution due to an improper set of initial weight vectors.

3·8 Discussion

In the introduction of this chapter it was stated that in this chapter the theory of biologically inspired pattern recognition systems, would be presented. After reading, this might be surprising because biology was hardly mentioned. The use of a linear discriminant as building block in the general architecture of a pattern classifier is probably the most obvious choice. The linear classification machine of figure 3·1 does not seem to have much relation to any biological system.

Table 3·3: Simple perceptrons and their properties.

Type	Data requirements	Decision regions	Convergence properties	Asymptotic properties
TLU	Linear dichotomy	Convex regions bounded by half-planes	Convergence	Unbiased
TLU+Gallant		Convex regions bounded by half-planes	Probabilistic convergence	Unbiased
Linear perceptron		Convex regions bounded by half-planes	SSE convergence	Mostly biased
Non-linear perceptron		Convex regions bounded by half-planes	Local SSE convergence	Approximately unbiased
Functional machine		Convex and non-convex regions of general shape	Convergence	Unbiased
Committee machine	Piece-wise linear dichotomy	Regions of any convexity bounded by half-planes	No guaranteed convergence	Unbiased

There is, however, a relationship to nervous systems. The on-off threshold device (for example the TLU) was a simplified model of the neuron and the main proponents of this idea were McCulloch and Pitts [McCulloch 1943]. The second concept was that long-term memory in animals depended on the changes in the synaptic junctions (the weights) between neurons, of which Hebb [Hebb 1949] was the main proponent. The

successful synthesis of these two ideas into the TLU is credited to Rosenblatt [Rosenblatt 1962].

The TLU or networks of these units are highly criticized as being plausible models of the nervous system [Crick 1989]. In this thesis, however, the biological aspects are not the main interest; the pattern recognition capabilities are the primary concern. It is interesting to observe the increase in capabilities, from TLU to committee machine. Unfortunately with an increase in complexity the theoretical foundations become weaker or undesirable properties emerge. In table 3·3 the properties and capabilities of the classifiers found so far are summarized.

Minsky and Papert [Minsky 1969] made clear that the complexity of perceptrons was too limited and that the research should be focused on how to create more complex systems. The increase in complexity can be achieved by adding functional links to the inputs but this requires full knowledge of the pattern probability distributions.

The alternative approach that uses TLU's as building blocks to create complex networks of perceptrons is still a non-parametric approach and therefore attractive from a pattern recognition point of view. A proper training algorithm was not known at the time Minsky and Papert published their book so research in this field stopped for some time. The following chapter will focus on layered networks of perceptrons and how to train such systems.

4
Multi-Layer Perceptrons

4·1 Introduction

The limitations of the simple perceptrons do not apply to the layered machines of perceptrons, that were introduced at the end of the previous chapter. This was already realized before the publication of Minsky and Papert [Minsky 1969] but only recently it has been shown how to train such systems. The committee machine exploits only a fraction of the true capabilities of layered machines, the weights connecting the hidden units with the output unit being fixed to one. A more serious problem is that the training procedure proposed in 3·7·3 is not guaranteed to converge.

The solution of the training problem is remarkable simple, it is a direct application of knowledge that was already present in the 1960's. The training is formulated as the optimization of a criterion function, as proposed in section 3·3·3. The difficulties imposed by the threshold elements are avoided by replacing these elements by bounded monotonic increasing functions such as 3·48 or 3·49. The gradient descent rule can now be applied, if the criterion function is differentiable, to find the weights.

This solution was invented several times in history, by [Bryson 1969], [Werbos 1974], [Parker 1985] and [Le Cun 1986]. The publication by [Rumelhart 1986], although not the first, gets most credit because it made the multi-layer perceptron popular.

In this chapter the basic theory and notations of multi-layer perceptrons will be discussed. In the literature a number of publications investigate the limitations of these new perceptrons, which are so general that they are often called *universal approximators*.

Besides the training technique described by [Rumelhart 1986], a number of alternatives are investigated as to their suitability for training these systems that should be capable of handling the large number of parameters that are often found in multi-layer perceptrons. Most of these alternative techniques are direct applications of known optimization methods to the network training.

A clear interpretation of network training, similar to perceptron training, is not possible any more but some asymptotic results are discussed. For example the layered structure of these networks is shown to perform some form of feature extraction. Finally recent developments and enhancements to the network architecture and training are discussed that are interesting from a pattern recognition point of view.

4·2 The general architecture

4·2·1 The feed forward architecture

The multi-layer perceptron is a layered machine that uses non-linear perceptrons as building blocks (see section 3·5). The output function of the non-linear perceptron is normally chosen to be a bounded monotonic increasing function and 3·49 is a popular choice. From a mathematical point of view this is not needed, however, as will be seen in section 4·5. A bounded monotonic increasing function is called a *squashing* function if the following limits hold:

$$\lim_{x\to\infty} f(x) = 1 \text{ and } \lim_{x\to-\infty} f(x) = 0 \qquad (4\cdot1)$$

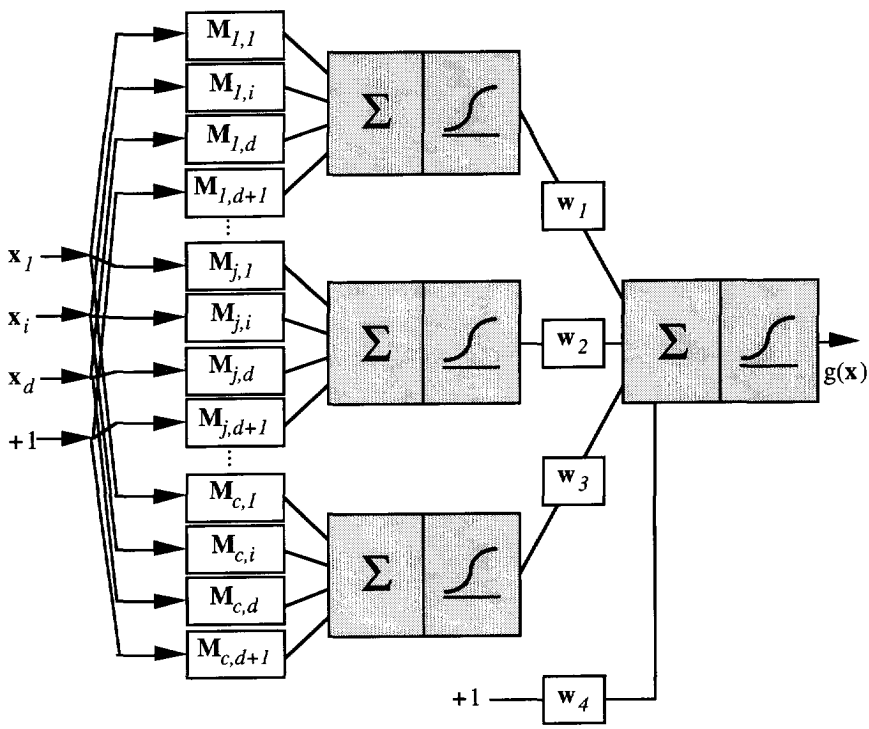

Figure 4·1: A feed forward network with three units in the hidden layer.

The units are organized in layers, of which the first layer receives as inputs the pattern to be classified. The outputs of this first layer are used as inputs to the second layer. The second and subsequent layers have as their inputs the outputs of the previous layer(s). The inputs of a unit in a hidden layer are not restricted to the outputs of the previous layers only, it may also receive outputs from earlier layers, for example, the pattern features of the first layer.

In the feed forward architecture the units are, however, not permitted to receive as input the outputs of units in any subsequent layer. There is therefore no feed back found in the system. The outputs of the units in the last layer are the response of the machine that can contain more than one unit in that layer, creating a multiple valued output machine.

This machine is less restricted than the committee machine. It may contain any number of hidden layers and any number of units per layer. A common convention to characterize a specific architecture is the following notation: $d\text{-}h_1..h_i..h_j\text{-}c$ In this notation d is the number of inputs, the pattern dimension and c is the number of outputs, the number of pattern classes. The numbers h_i represents a layer with that number of units in the layer. In figure 4·1 an example of a *d-3-1* feed forward network is shown.

Most applied networks have only one hidden layer, an architecture that is (in theory) complex enough to solve most problems (see section 4·2·2). In mathematical formulation one hidden layer network is given by:

$$g_i(\mathbf{y}) = f(\sum_{j=1}^{h} \mathbf{W}_{i,j}\, f(\mathbf{M}_j \cdot \mathbf{y}) + \mathbf{W}_{i,h+1}) \tag{4·2}$$

In this formula all weights connecting the inputs with the units in the hidden layer are contained in matrix $\mathbf{M}$. The matrix $\mathbf{W}$ holds all weights of the units in the output layer. The pattern vector $\mathbf{y}$ is again an augmented pattern vector. The function $g_i(\mathbf{y})$ represents the i-th output of the network and a network with c outputs is thus characterized by c of these functions.

The set of functions $g_i(\mathbf{y})$ can be interpreted as discriminant functions, similar to the discriminant functions found in the general pattern classifier of section 2·2·3. The network shown in figure 4·1 is a detailed view of the black boxes in figure 2·2. A feed forward network that is used as a pattern classifier therefore needs a maximum device to determine the network response. The simplification for the two-category case, to use one discriminant function instead of two, results in a network with one output. This output is thresholded at the appropriate level to determine the network classification.

It should be noted that *all* functions $g_i(\mathbf{y})$ are dependent on matrix $\mathbf{M}$, which means that the discriminant functions share a common data transformation. If the matrix $\mathbf{M}$ is fixed, the feed forward network is no more than a set of c non-linear perceptrons. These perceptrons classify a sample using a new data set of which the features are non-linear functions of the original data. These new features, given by the transformation $f(\mathbf{M}_i \cdot \mathbf{y})$, are called the *hidden units' activations.*

An obvious simplification of this architecture is to remove the output functions in the last layer (making them linear), resulting in the following networks:

$$g_i(\mathbf{y}) = \sum_{j=1}^{h} \mathbf{W}_{i,j}\, f(\mathbf{M}_j \cdot \mathbf{y}) + \mathbf{W}_{i,h+1} \tag{4·3}$$

Clearly the capability of this last network is equal to the networks given by 4·2 because f() is a squashing function. The weights contained in matrix $\mathbf{W}$ of this last architecture, can be optimized with the matrix inverse technique of equation 3·31, if the

matrix **M** is fixed. The pattern error function, however, can result in a classifier with higher classification error as discussed in section 3·5·2.

4·2·2 Feed forward networks and universal approximation

In the classical paper by Rumelhart, Hinton and Williams [Rumelhart 1986] the authors immediately investigated the capabilities of networks given by equation 4·2. They show that these structures can solve some of the problems formulated by Minsky and Papert [Minsky 1969]. In [Rumelhart 1986], it is shown that the proposed training algorithm, the *back propagation method* (see 4·3·1), can find a solution for the exclusive-or and parity problem, mentioned in section 3·7·1.

The parity problem and five other applications inspired by biology and engineering are demonstrated to be solved adequately with feed forward networks trained with the back propagation algorithm. The authors ([Rumelhart 1986, page 695]) concluded from these results that they believe to have answered Minsky and Papert's challenge: *"In short, we believe that we have found a learning result sufficiently powerful to demonstrate that their pessimism about learning in multi-layer machines was misplaced."*

This conclusion is purely based on an experimental observation and the theoretical limitation of discriminant functions 4·2 and 4·3, if they can be formulated, is an important and interesting question to be answered.

The functions 4·2 and 4·3, implemented by the feed forward network, can be used as discriminant functions, but also to approximate any mapping of the form $g_i(\mathbf{y})$: $\Re^{d+1} \rightarrow \Re$. The difference between these two situations is subtle, in the first case the desired response or target t_y is a fixed value for every pattern of the same class (see for example equation 3·46). In the second case the target can have any value as function of the input **y**.

A more general question about the capabilities of functions 4·2 and 4·3 is how well they can approximate any measurable function. This question is closely related to Hilbert's 13th open problem as he formulated that at the 2nd International Congress of Mathematics held in Paris 1900 [Kurkova 1992, page 502]. Although Hilbert's hypothesis was already disproved by [Kolmogorov 1957], a more general statement about approximating functions by sums of one variable continuous functions is given by [Lorentz 1966]. The capabilities of functions 4·2 and 4·3 are reformulations of the theorems by Kolmogorov and Lorents for neural networks.

In [Funahashi 1989], [Hornik 1989] and [Kurkova 1992] it is claimed that feed forward networks with one hidden layer are universal approximators, which means that they can approximate any measurable function from one finite dimensional space to another, to any degree of accuracy, provided that enough hidden units are available. Stated more formally, the approximation capabilities of the following restricted family of functions is given by the next theorem:

Theorem 4·1: Universal approximation theorem [Hornik 1989, page 362]: Let the class of functions g() be given by equation 4·4 where f() is a squashing function. This class of functions can approximate any continuous function uniformly on any compact set, regardless of the squashing function and regardless of the input space environment. The compact set requirement holds whenever the possible values of the input space **y** are bounded.

$$g(\mathbf{y}) = \sum_{j=1}^{h} \mathbf{w}_j \, f(\mathbf{M}_j \cdot \mathbf{y}) \tag{4·4}$$

For a finite training set the number of hidden units h needed to approximate a desired unknown function g'() is bounded by a theorem from [Hornik 1989].

Theorem 4·2: The maximum of number of hidden units [Hornik 1989, page 362]. Let D be a finite design set with m distinct points. Let g'(**y**): $\Re^{d+1} \rightarrow \Re$ be an arbitrary function to be approximated, than there exists a function g(**y**) from the family given by equation 4·4, with $h=m$ hidden units such that g(**y**)=g'(**y**) **y**$\in D$.

The universal approximation property shows that feed forward networks form a powerful set of (discriminant) functions. If this result is applied to a network used as pattern classifier, any continuous decision boundary can in theory be approximated. Of course this result must be carefully interpreted.

The upper bound on the required number of hidden units can grow exponentially [Hertz 1991, page 142] with the pattern dimension, depending on the complexity of the data. This may seem unrealistic if some compactness in input space exists, however, the theorems do not give any clue about a scaling relationship between input complexity and the number of hidden units. It is therefore useful to know which functions can efficiently be represented by feed forward networks.

The architecture may in theory be capable of approximating any desired discriminant, it is not known if the training algorithm, used to adjust the weights, is capable of finding a solution. The true network capabilities are dependent both on the architecture and training algorithm. In the next section different training algorithms will be discussed.

The network capabilities of equations 4·2 and 4·3 as classifiers can also be investigated by applying the results from Cover, discussed in section 3·6·1. The weights of matrix **W** in equation 4·2 and 4·3 implement a non-linear and linear perceptron respectively. The hidden unit activations f($\mathbf{M}_i \cdot \mathbf{y}$) form the features for these perceptrons. A new data set can be constructed from the original data, that are the patterns constructed from the hidden units activation's. In formula:

$$\mathbf{y}' = (f(\mathbf{M}_1 \cdot \mathbf{y}), \ldots, f(\mathbf{M}_i \cdot \mathbf{y}), \ldots, f(\mathbf{M}_h \cdot \mathbf{y}), 1)^T \tag{4·5}$$

The matrix **M** can be chosen such that the new patterns **y'** are in a general position. The linear or non-linear perceptron at the output can arbitrarily label $h+1$ samples (the

Cover result see section 3·6·1). From this result it can be concluded that the feed forward architecture is at *least* capable of correctly classifying $h+1$ samples. This agrees with theorem 4·2 because the networks described by equation 4·4 do not contain the trainable threshold element and thereby have a capacity that is one less.

If it is assumed that the training will result in a *random* positioning of the new patterns $\mathbf{y}'$ in the $h+1$ dimensional space, the following suggestion can be made about the minimal number of training patterns to be used. Following Cover's result the number of samples used to train a one hidden layer network should be chosen such that:

$$m > 2(h+1) \tag{4·6}$$

This sample size is not a sufficient condition to ensure generalization, but should be interpreted as an absolute minimum sample size to train a feed forward classifier.

4·3 Training feed forward networks

4·3·1 The back propagation method

The back propagation method proposed by [Rumelhart 1986] is probably the most widely used method to train feed forward networks. The training of a feed forward network is defined as the minimization of the sum of squared errors criterion:

$$J_{sse}(\mathbf{W},\mathbf{M}) = \sum_{\mathbf{y}\in D}\sum_{i=1}^{c}(\mathbf{t}_{\mathbf{y},i} - g_i(\mathbf{y}))^2 \tag{4·7}$$

In this formula $\mathbf{t}_{\mathbf{y},i}$ is the i-th component of the target vector corresponding to pattern $\mathbf{y}$. In classification this vector is usually a fixed vector for all samples of the same class. A design set D is now a set of training patterns, with corresponding target *vectors*. The steepest descent rule of 3·15 is applied to 4·7. In this case the chain rule of differentiation must be applied that results in more complicated formulas than for the non-linear perceptron (3·52). Applying this to the *one* hidden layer network of 4·2 the update rule for the *output* units becomes:

$$\mathbf{W}'_i = \mathbf{W}_i + \eta\sum_{\mathbf{y}\in D}(\mathbf{t}_{\mathbf{y},i} - g_i(\mathbf{y}))f'(\mathbf{W}_i\cdot\mathbf{y}')\mathbf{y}' \tag{4·8}$$

In this formula $\mathbf{y}'$ is the vector of inputs to the output unit, that is, in a one hidden layer network equal to the vector given by 4·5. This rule is equal to the update rule of the non-linear perceptron 3·52, if this is compared to training such a perceptron with the transformed patterns $\mathbf{y}'$.

The update rule for the *hidden* units follows from the chain rule of differentiation and for a one hidden layer network is given by:

$$\mathbf{M}'_i = \mathbf{M}_i + \eta\sum_{\mathbf{y}\in D}([\sum_{j=1}^{c}\mathbf{W}_{i,j}(\mathbf{t}_{\mathbf{y},j} - g_j(\mathbf{y}))f'(\mathbf{W}_j\cdot\mathbf{y}')]f'(\mathbf{M}_i\cdot\mathbf{y})\mathbf{y}) \tag{4·9}$$

A clever simplification and generalization of these formulas to feed forward networks with more hidden layers is given in [Rumelhart 1986, page 679], where the common structure of equations 4·8 and 4·9 is shown. The update rule, in terms of an individual unit with weight vector **w** and input vector **y**, can be formulated as:

$$\mathbf{w}' = \mathbf{w} + \eta \sum_{\mathbf{y} \in D} \Delta_{\mathbf{y}} \mathbf{y} \tag{4·10}$$

The quantity $\Delta_{\mathbf{y}}$ for the i-th output unit is:

$$\Delta_{\mathbf{y},i} = (\mathbf{t}_{\mathbf{y},i} - g_i(\mathbf{y}))\mathrm{f}'(\mathbf{W}_i \cdot \mathbf{y}') \tag{4·11}$$

If equations 4·10 and 4·11 are combined, the update rule 4·8 for the output units is obtained. The quantity $\Delta_{\mathbf{y}}$ for the i-th hidden unit is in this formulation:

$$\Delta_{\mathbf{y},i} = (\sum_{j=1}^{c} \mathbf{W}_{i,j} \Delta_{\mathbf{y},j}) \mathrm{f}'(\mathbf{M}_i \cdot \mathbf{y}) \tag{4·12}$$

If again equations 4·10 and 4·12 are combined, the hidden unit update rule of 4·9 will be obtained. These results generalize to a feed forward network with any number of hidden layers. In this case both 4·10 and 4·11 remain unchanged. Equation 4·12 can still be applied if it is realized that $\mathbf{M}_i$ is the weight vector of the current unit, $\mathbf{W}_{i,j}$ is the weight between the i-th unit in the current layer and the j-th unit in the next layer, and that $\Delta_{\mathbf{y},j}$ is the delta of the j-th unit in the next layer.

The stochastic approximation of this rule is called the *generalized delta rule*, because it is the generalization of the Widrow-Hoff delta rule to the feed forward architecture. The generalized delta rule is just the sample update version of the steepest descent rule, so equation 4·10 becomes:

$$\mathbf{w}' = \mathbf{w} + \eta \Delta_{\mathbf{y}} \mathbf{y} \tag{4·13}$$

The application of the generalized delta rule involves two phases. In the first phase the pattern is presented to the network and propagates forward through the network to compute all output values and activations. These outputs are compared with the targets, resulting in a delta $\Delta_{\mathbf{y},j}$ at each output unit (equation 4·11). The second phase involves a backward pass through the network during which the error is passed to each unit in the network (equation 4·12) and the appropriate weight changes (equation 4·13) are made. The backward pass of this algorithm inspired the popular name of this algorithm, which is known as the *back propagation algorithm.*

It is interesting that a number of modifications can be made to the network architecture without effecting the training algorithm. Any number of weights can be fixed in the network. In this case the error is back propagated as before only the fixed weights are simply not changed. Furthermore there is no reason why some units might not receive as inputs, the outputs of units in any earlier layer. In this case, the errors are simply back propagated along these connections.

For the true gradient descent version of this algorithm, the weight changes are accumulated for the complete design set and the weights are changed (equation 4·10)

only when the complete set is passed. In [Rumelhart 1986, page 680] it is noted that the sample update learning is critical to the learning rate η and the following modified steepest descent is proposed to increase robustness and convergence speed:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \nabla J(\mathbf{w}^{(i)}) + \alpha(\mathbf{w}^{(i)} - \mathbf{w}^{(i-1)}) \tag{4·14}$$

The term multiplied by α is just the previous weight update, so any positive value for this parameter determines the effect of the past weight change on the current step in weight space. The parameter α is called the *momentum* term and effectively filters out some high frequency variations of the error surface in the weight space. The implementation of this modification requires the storage of the previous weight update for every parameter in the network.

One last problem is how to choose the initial weights. If all weights are chosen equal, zero for example, then all the errors propagated back through the network will be equal. Because the weight changes are dependent on this error, all units will be updated in the same way thereby keeping all weights equal. If the solution requires that unequal weights must be developed, then equal initial weights will never lead to a solution. In [Rumelhart 1986, page 681] it is proposed to use small random weights as initial weights to avoid this problem.

4·3·2 Stability and convergence properties

The stability and convergence properties of the steepest descent with momentum (4·14) differ from the original steepest descent 3·15. The convergence properties will be investigated by examining the properties for the linear perceptron. The theory in this section closely follows the approach of section 3·4·3.

The error space of the linear perceptron is a quadratic function, as given by equation 3·32. The stability and convergence will first be investigated for the one dimensional weight space 3·33. If the modified steepest descent rule 4·14 is applied to 3·33 then the following second order difference equation is obtained:

$$w^{i+1} = w^i - 2\eta\lambda(w^i - w^*) + \alpha(w^i - w^{i-1}) \tag{4·15}$$

or

$$(w^{i+1} - w^*) - (1 + \alpha - 2\eta\lambda)(w^i - w^*) + \alpha(w^{i-1} - w^*) = 0 \tag{4·16}$$

The stability of this difference equation follows from linear system theory ([Oppenheim 1983, page 655]). The system described by 4·16 is stable if the roots r of the characteristic function are in the unit circle:

$$r^2 - (1 + \alpha - 2\eta\lambda)r + \alpha = 0 \tag{4·17}$$

The root locus can be plotted by fixing α and varying η. In figure 4·2 the root locus, obtained from [Tugay 1989, page 118], is shown. From this figure it is clear the modified steepest descent is stable if:

$$-1 < \alpha < 1 \text{ and } 0 < \eta < \frac{1+\alpha}{\lambda} \tag{4·18}$$

The vector equivalent of 4·16 is more complicated. First the original **w** space is translated to a space with **w*** as origin and second this system is rotated to a new axis, given by matrix **Q** of equation 3·39. In this new coordinate system the following uncoupled vector equation can be derived [Tugay 1989, page 118]:

$$\mathbf{w}^{i+1} - (\mathbf{1} + \alpha\mathbf{1} - 2\eta\Lambda)\mathbf{w}^{i} + \alpha\mathbf{w}^{i-1} = 0 \tag{4·19}$$

Here Λ is the diagonal matrix of equation 3·39 and this system is stable if the roots of a set of equations equal to 4·17, are collectively inside the unit circle. Since η is fixed for all parameters, the necessary condition for convergence is:

$$-1 < \alpha < 1 \text{ and } 0 < \eta < \frac{1+\alpha}{\lambda_{max}} \tag{4·20}$$

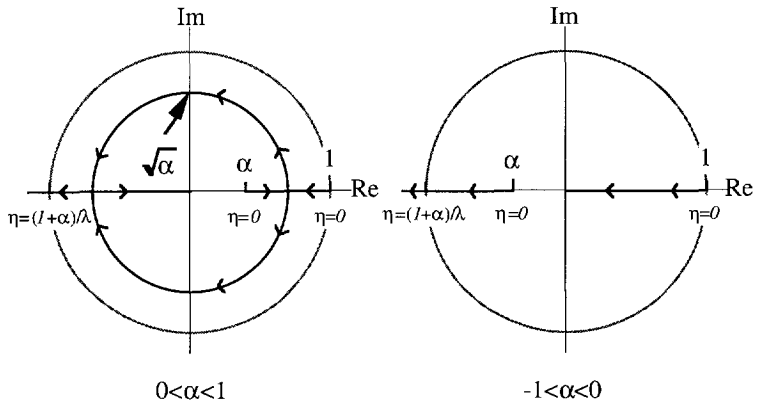

Figure 4·2: The root locus of the steepest descent algorithm with momentum.

For the non-linear perceptron and feed forward network this convergence condition is not sufficient any more. The local dynamics of the training procedure can be analyzed by approximating 3·50, 4·2 or 4·3 by a first order Taylor series in **w**. Stability of the system for every possible series expansion in **w**, is identical to the absolute stability criterion found in numerical integration as noted before in section 3·5·1.

In practice a learning rate η and momentum α for the training of a feed forward network or non-linear perceptrons, will be found by trial and error. The user should do this careful, because a combination of η and α can provide a stable result for a particular random initial weight vector $\mathbf{w}^{(0)}$. A second try with different initial weights, but all other settings kept equal, can result in unstable learning. The explanation for this observation is that training sessions follow different paths in weight space, with the possibility that the first path is smooth and may learn in a stable way, but a second more rough path can lead to unstable learning. The same phenomenon can be observed if the

stochastic approximation (4·13) is used and the order of presentation of the samples is different.

4·3·3 Other iterative training methods

In practice the training of a feed forward network is difficult and most times terribly slow. The networks are often complex, involving many units and interconnections. The optimization problem, as it is formulated by formula 4·7, is of high dimension. The number of trainable parameters N rapidly increases with an increasing number of hidden units. The non-linearity of the units constrains the choice of an algorithm to solve the learning problem.

In the literature a large number of papers have been published on the problem of training feed forward networks, especially to improve convergence speed. Three examples are [Silva 1990], [Stornetta 1987] and [Jacobs 1988]. In the past thirty years the subject of numerical techniques for non-linear optimization has been extensively researched. Many optimization techniques have been investigated and a number of these techniques can be applied to network training. It is interesting which of those techniques could be used specifically to improve training speed and robustness.

In this literature it is already known for long time that in most non-linear least squares problems, the minima are located in elongated, curved valleys, often referred to as banana shaped valleys (see [Van den Bos 1982, page 349]). Close to a minimum, the steepest descent proceeds along a zigzag path and converges slowly. Far from the minimum, however, the steepest descent is fast.

An optimization technique that converges faster near the minimum is the Newton's algorithm that uses second-order information. The Newton's update rule is:

$$\mathbf{w}' = \mathbf{w} - \eta \left(\frac{\partial^2 \mathrm{J}(\mathbf{w})}{\partial \mathbf{w}^2} \right)^{-1} \nabla \mathrm{J}(\mathbf{w}) \tag{4·20}$$

The computation of this method is very expensive because the inverse of the Hessian matrix must be computed. The Hessian matrix is complicated to calculate (see [Buntine 1991]). The size of this square matrix is NxN, where N is the number of trainable parameters, which is large ($N>100$) as discussed above. The computation of the inverse of this matrix is of order N^3 which is even more dramatic.

The computation of the Hessian matrix can be avoided by approximating this matrix. This results in the Gauss-Newton iteration scheme ([Van den Bos 1982, page 350]):

$$\mathbf{w}' = \mathbf{w} - \frac{1}{2} \left(\frac{\partial \mathbf{g}^T}{\partial \mathbf{w}} \cdot \frac{\partial \mathbf{g}}{\partial \mathbf{w}} \right)^{-1} \nabla \mathrm{J}(\mathbf{w}) \tag{4·21}$$

In this last formula, vector $\mathbf{g}$ is given by:

$$\mathbf{g} = (g(\mathbf{y}_1), \ldots, g(\mathbf{y}_i), \ldots, g(\mathbf{y}_m))^T \tag{4·22}$$

Two properties ([Van den Bos 1982, page 350]) of the Gauss-Newton method are the rapid convergence close to the minimum, but a divergence far from the minimum. The approximation of the Hessian matrix is only accurate near the minimum. Far from the minimum this is not valid any more and the approximation can even become singular.

Comparing the properties of steepest descent and Gauss-Newton suggests combining both methods into one method. This is known as the *Marquardt* method ([Van den Bos 1982, page 350]), given by the following update rule:

$$\mathbf{w}' = \mathbf{w} - \frac{1}{2}\left(\frac{\partial \mathbf{g}^T}{\partial \mathbf{w}} \cdot \frac{\partial \mathbf{g}}{\partial \mathbf{w}} + \eta \mathbf{1}\right)^{-1} \nabla J(\mathbf{w}) \tag{4·23}$$

The parameter η in 4·23 determines if the algorithm behaves as steepest descent or Gauss-Newton step. The parameter η is initially chosen large, compared to Gauss-Newton approximation, resulting in a steepest descent step. When a minimum is approached, η is reduced so that the algorithm is a Gauss-Newton step. The technique to optimize η during the optimization process can be found in ([Van den Bos 1982, page 351]).

The Marquardt method proved to work very well in practice ([Van den Bos1982, page 351]) and has become a standard in numerical optimization. This method is included in a number of scientific program libraries .

The Gauss-Newton and Marquardt methods approximate the Hessian but still require a matrix inverse. The quasi-Newton methods were developed, in which the inverse Hessian is successively approximated. This avoids the direct computation of second order derivatives and the computation is reduced to a complexity of N^2.

A popular equation is the so called *Davidon-Fletcher-Powell* algorithm (see for an introduction [Press 1988, section 10·7]). The algorithm and the iterative approximation $\mathbf{H}$ of the inverse of the Hessian are given by:

$$\begin{cases} \mathbf{w}^{i+1} = \mathbf{w}^i - \eta \mathbf{H}^i \nabla J(\mathbf{w}^i) \\ \mathbf{H}^{(i)} = \mathbf{H}^{(i-1)} + \dfrac{[\mathbf{w}^{(i)} - \mathbf{w}^{(i-1)}][\mathbf{w}^{(i)} - \mathbf{w}^{(i-1)}]^T}{[\mathbf{w}^{(i)} - \mathbf{w}^{(i-1)}]^T [\nabla J(\mathbf{w}^{(i)}) - \nabla J(\mathbf{w}^{(i-1)})]} \\ \qquad - \dfrac{\mathbf{H}^{(i-1)}[\nabla J(\mathbf{w}^{(i)}) - \nabla J(\mathbf{w}^{(i-1)})][\nabla J(\mathbf{w}^{(i)}) - \nabla J(\mathbf{w}^{(i-1)})]^T \mathbf{H}^{(i-1)}}{[\nabla J(\mathbf{w}^{(i)}) - \nabla J(\mathbf{w}^{(i-1)})]^T \mathbf{H}^{(i-1)} [\nabla J(\mathbf{w}^{(i)}) - \nabla J(\mathbf{w}^{(i-1)})]} \end{cases} \tag{4·24}$$

The initial matrix $\mathbf{H}^{(0)}$ is set to the unit matrix and the parameter η is optimized by performing a line search. A related algorithm by the name *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) is often implemented in numerical packages. This method differs only in details of round-off error and is computationally slightly more complicated, see [Press 1988, page 324-326].

A method that does *not* explicitly store the inverse of the Hessian matrix, but which explores its properties during the iterations, is the *conjugate gradient descent* method.

This method is attractive for the optimization of feed forward networks because it only stores gradient information and it is therefore of order N. The underlying principles of this method are complicated and a good introduction can be found in [Press 1988, section 10·6]. The weight update is given by the two equations:

$$\begin{cases} \mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \eta \mathbf{j}^{(i)} \\ \mathbf{j}^{(i)} = -\nabla J(\mathbf{w}^{(i)}) + \dfrac{(\nabla J(\mathbf{w}^{(i)}) - \nabla J(\mathbf{w}^{(i-1)}))^T \nabla J(\mathbf{w}^{(i)})}{|\nabla J(\mathbf{w}^{(i-1)})|^2} \mathbf{j}^{(i-1)} \end{cases} \tag{4·25}$$

In the first iteration, the search direction $\mathbf{j}$ is a simple steepest descent:

$$\mathbf{j}^{(0)} = -\nabla J(\mathbf{w}^{(0)}) \tag{4·26}$$

The conjugate gradient descent has advantages in both speed and convergence over steepest descent. The computations only involve gradients that can efficiently be calculated with the back propagation rule 4·10. Furthermore the learning rate η is optimized using a line search during training (see [Press 1988, section 10·6]), so this method does not need any user interaction to find a learning rate that leads to stable learning.

While watching the training of the weights in a feed forward network as a function of time (iteration number), Owens and Filkin [Owens 1989] noticed similarities with differential equation systems that are stiff. Stiff differential equations are characterized by time constants that vary greatly in magnitude or eigenvalues for which the fraction $\lambda_{max} / \lambda_{min}$ is large. In back propagation training this corresponds to the phenomenon that some weights reach early steady state while others only vary slowly. Some weights in training remain unchanged for a large number of iterations and then suddenly change to another quasi steady state. This behavior is common to stiff differential systems.

Mathematicians working in the field of numerical integration, struggled for a long time to find efficient algorithms for stiff differential equations. Owens and Filkin [Owens 1989] proposed rewriting the steepest descent as a system of coupled ordinary differential equations and using these algorithms to train a multi-layer perceptron. In steepest descent the change of the weights in time is proportional to minus the gradient of the objective function. This suggests that the steepest descent (3·15) can also be formulated as follows:

$$\frac{\partial \mathbf{w}}{\partial t} = -\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \tag{4·27}$$

The discrete difference equation of 3·15 is replaced by a set of ordinary differential equations. The fixed step Euler method [Gear 1971, page 10] for integrating differential equations applied to 4·27 results in equation 3·15.

The stiffness of the solution of 4·27 (see [Gear 1971, page 209]) is determined by the Hessian of the optimized criterion function J(). The ratio of the largest and smallest eigenvalues, the condition number of this matrix, determines the stiffness. A similar discussion for the steepest descent was found in section 3·4·3 but extends to differential

equations. This condition number limits the training step and thereby the convergence speed.

Systems with very large number of parameters are very likely to be stiff. The condition number of a N by N dimensional matrix, that is bordered with an N-dimensional vector and a new diagonal element, will at best remain equal. In most situations, however, where correlation exists between the new and old parameters, the condition number will become larger. The mathematical proof and consequences of this theory for training networks can be found in [Bakker 1993]. The conclusion from this theory is that when the complexity of a system is increased, it becomes more probable that it is ill-conditioned.

Feed forward networks often have a large number of trainable parameters, so stiffness is likely to limit network training. Owens and Filkin [Owens 1989] proposed using a method developed for stiff differential equations to solve the system of equations 4·27. The initial values are chosen to be small random weights and the steady state values correspond to the minimum sum of squared errors solution. The method of Gear is recognized as a well-tested method to solve stiff differential equations [Gear 1971]. Unfortunately it is of complexity N^3 because it involves a matrix inverse.

4·3·4 Efficiency of training methods

The training algorithms discussed in the sections 4·3·1 and 4·3·3 differ from order N to N^3 in the complexity of one iteration. The effectiveness of a training method is also dependent on the number of iterations needed to reach a certain solution. A method of computational complexity N^2 can still be faster than a method of order N, if it needs less than $1/N$ iterations. A method of higher complexity is likely to need fewer iterations, because it uses more information to perform a step in weight space.

Table 4·1: Summary of iterative training methods.

Abbreviation	Method	Equation	Complexity
SD	Steepest descent	4·10	N
BP	Generalized delta rule	4·13	N
MAR	Marquardt	4·23	N^3
BFGS	Pseudo Newton (BFGS)	4·24+[Press 1988, page 326]	N^2
CG	Conjugate gradient descent	4·25	N
STIFF	Stiff differential equations	4·27+[Gear 1971, page 209]	N^3

It is difficult to predict which method will be faster in computing a solution. The effectiveness of one iteration is problem specific. The advantage of a more complex technique might be greater for small or moderate sized problems. Besides the computational complexity, these methods differ in storage requirements. The storage of an N by N matrix, for the BFGS algorithm for example, limits the applicability for large networks.

A number of experiments have been performed to compare the properties of the six training techniques discussed so far. These six methods are listed in table 4·1, together with the complexity and fundamental learning equation.

The training techniques are compared for three artificially generated data sets. All the data sets have two dimensional *Gaussian* distributed samples with two different classes. The first data set is the Highleyman set of which the statistics are shown in section 3·5·3 (equation 3·55). The statistics of the two other data sets are given by the following means and covariances:

$$\begin{aligned} \omega_1 &: \ \bar{\mathbf{x}}_{\omega_1} = (0.2,\ 0.2)^T, \quad \mathbf{S}_{\omega_1} = \begin{pmatrix} 0.111 & 0 \\ 0 & 0.111 \end{pmatrix} \\ \omega_2 &: \ \bar{\mathbf{x}}_{\omega_2} = (-0.2,\ -0.2)^T, \quad \mathbf{S}_{\omega_2} = \begin{pmatrix} 0.111 & 0 \\ 0 & 0.111 \end{pmatrix} \end{aligned} \tag{4·28}$$

$$\begin{aligned} \omega_1 &: \ \bar{\mathbf{x}}_{\omega_1} = (0,\ 0)^T, \quad \mathbf{S}_{\omega_1} = \begin{pmatrix} 2.71 & 0 \\ 0 & 2.71 \end{pmatrix} \\ \omega_2 &: \ \bar{\mathbf{x}}_{\omega_2} = (0,\ 0)^T, \quad \mathbf{S}_{\omega_2} = \begin{pmatrix} 0.369 & 0 \\ 0 & 0.369 \end{pmatrix} \end{aligned} \tag{4·29}$$

For both data sets the *a priori* probabilities of the classes are equal (50%) and the Bayes classifier has a corresponding error of $\varepsilon^* = 0.20$. The optimal classifier for 4·28 is linear (see figure 4·3) and for 4·29 a circle (see figure 4·4) with center at the origin and radius ≈ 1.85.

A training set of 200 samples (100 per class) is drawn for all three distributions. These three data sets, shown in figures 3·11, 4·3 and 4·4, are used to compare the six training algorithms.

The training algorithms are implemented in the programming language 'C' [Kernighan 1978] and are all part of the *artificial neural network library* ANNLIB (see chapter 8). The steepest descent and generalized delta rule are direct implementations of the algorithm as discussed in section 4·3·1. The conjugate gradient descent, BFGS and Marquardt algorithms make use of the corresponding routines listed in [Press 1988]. The gradients are calculated with equations 4·11 and 4·12. The stiff method uses a translated FORTRAN version of Gear's method ([Gear 1971]) and the right hand side of 4·27 is also calculated with equations 4·11 and 4·12.

To compare the learning methods, the effectiveness is measured as the amount of processing time needed on a Sun SparcStation1, to reach a solution for which $J_{sse}() < 2$. On average this corresponds to a network classifier with a re-substitution error of $\hat{\varepsilon}_r \approx 0.20$. The processing times are averaged for 8 different initial weight matrices (**M** and **W**) and are measured for different network architectures. The networks that are used all contain one hidden layer and one output (*2-h-1*). The number of hidden units h is varied between 2 and 128. In the figures showing the results the number of trainable parameters N is plotted, which is related to h as follows: $N = d \cdot h + 2 \cdot h + 1$.

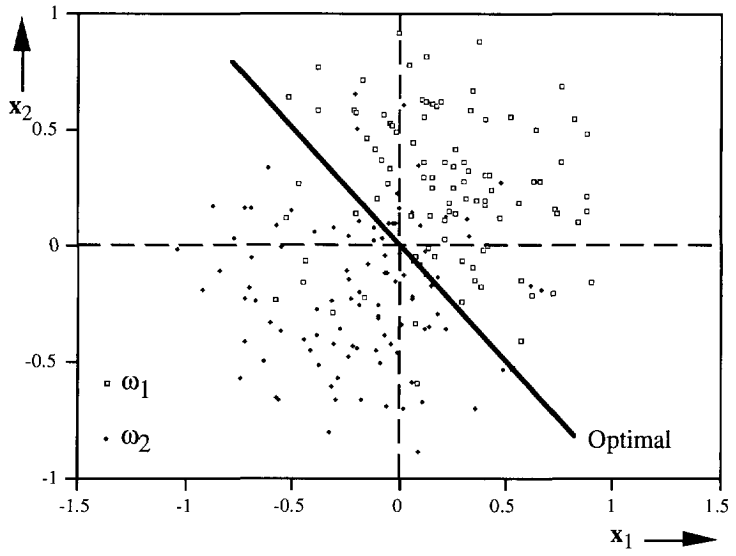

Figure 4·3: The training set of 200 samples for patterns distributed with Gaussian probabilities and statistics given by 4·28. The Bayes classifier is shown.

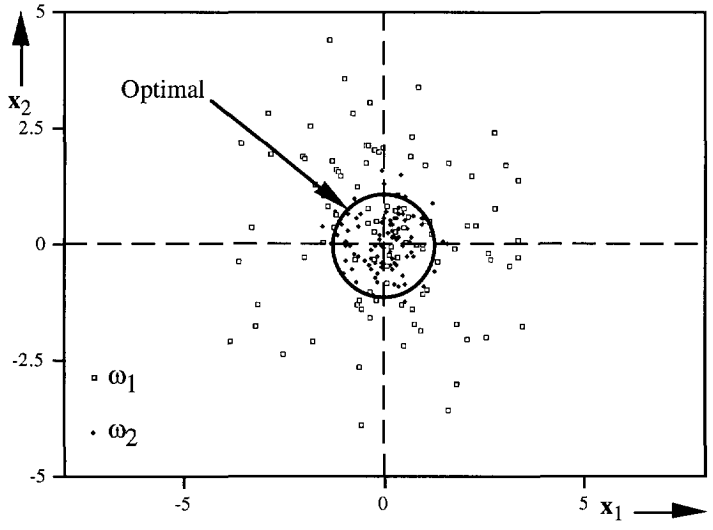

Figure 4·4: The training set of 200 samples for patterns distributed with Gaussian probabilities and statistics given by 4·29. The Bayes classifier is the circle with radius 1.85.

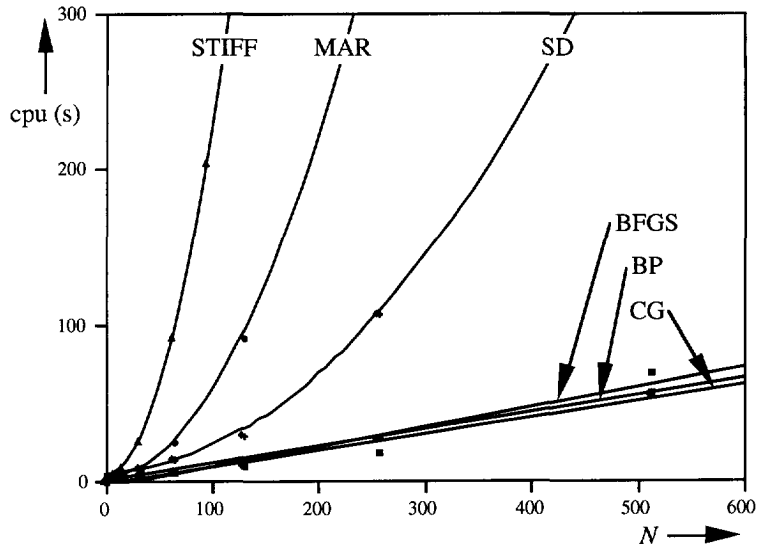

Figure 4·5: The computational complexity measured as function of the number of trainable parameters N for the Highleyman data set (3·55).

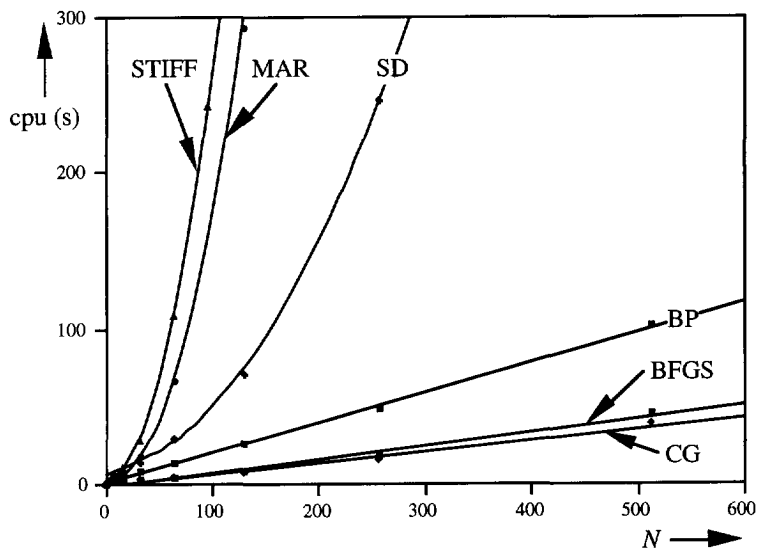

Figure 4·6: The computational complexity measured as function of the number of trainable parameters N for the data set with different means (4·28).

The generalized delta rule (BP) and steepest descent (SD) both use a fixed learning rate η, that was chosen such that all training sessions show stable learning in time. This number influences the effectiveness and thereby the conclusion, so these results should

be interpreted with care. One identical learning rate for all experiments was not possible because this lead to instabilities. In most cases η=0.1 and α=0.0 were chosen, except for data set 4·29 where η=0.05 and α=0.5 were used.

In figures 4·5, 4·6 and 4·7 the measured computational complexities for the three different data sets are shown. The stiff differential equation method, the Marquardt method and the steepest descent are the most computationally expensive for all three examples. The steepest descent is this implementation uses a fix step. A variable step implementation is expected to perform better, but is not compared here (see [Watrous 1987]). The matrix inverse that is part of Marquardt and the stiff method seems to limit its usefulness to only small size problems. This conclusion is supported by experiments reported in [Boer 1991].

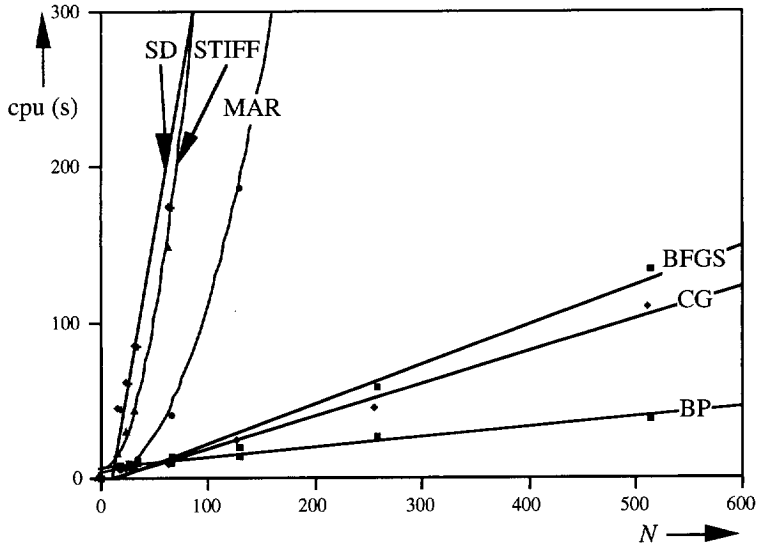

Figure 4·7: The computational complexity measured as function of the number of trainable parameters N for the data set with different covariances (4·29).

Back propagation, BFGS and conjugate gradient descent roughly scale linearly with the problem complexity. From these experiments not one of these three seems to be the best. The BFGS requires the storage of the Hessian inverse matrix making it less desirable for large size networks from a memory point of view.

The generalized delta rule (BP) needs user interaction to find a good set of training parameters; this can be difficult. The advantage of the conjugate gradient descent (CG) is simplicity and robustness. It requires only twice as much storage as the back propagation algorithm and optimizes a learning rate during the process.

An important aspect is that the conjugate gradient descent and the BFGS method have better convergence properties near a minimum. The effectiveness was measured as

the CPU time needed to reach a solution for which $J_{sse}() < 2$. It is possible that this target is far from a minimum, so that better convergence properties are simply not noticed. The dependencies between the target criterion and the improvement of BFGS and CG are shown in [Watrous 1987, page 626] and [Moller 1990, page 16]. In both articles the improvement of BFGS or CG increases when the target squared error is lowered and in their examples these techniques shown an improvement that is one order of magnitude.

For a feed forward network that is used as a classifier, it is not necessary to find a network solution with a sum of squared errors that is as low as possible. This will be discussed in chapter 5. The target value chosen for this experiment is a realistic choice because it corresponds to a classifier with a re-substitution error that is near the Bayes error. It is likely that the BFGS and CG methods are favorable for training networks that are used as function approximators where a solution with the lowest J_{sse} is desirable. In [Johansson 1990, page 19] it is indeed concluded that conjugate gradient descent is superior in speed and robustness to the back propagation method.

An important remark concerns the training set size m here chosen as 200. The generalized delta rule (BP) estimates the gradient using one sample, in contrast to the other methods. The evaluation of the derivatives and the value of the criterion function J() during the line minimization is more complex for large data sets than for small sets. This suggests that the generalized delta rule is faster for larger data sets than for smaller sets.

In conclusion, the choice of optimization technique is difficult and complex. The STIFF, MAR and SD methods appear to be computationally more complex than BP, BFGS and CG and are not considered to have favorable scaling properties. From these last three alternatives BP and CG are economical in space but which method is better is problem specific. This choice depends on the distribution of the data, the desired task and the training sample size. The conjugate gradient descent does not need any user interaction and is reported to learn in a stable and robust way ([Johansson 1990] and [Moller 1990]). This method should be considered if no specific knowledge of the application is available.

4·3·5 A non-iterative training technique

In this section a new and non-iterative method is proposed to compute the weights of a *single* hidden layer network. This method builds upon other non-iterative techniques and is motivated by previous work of [Haersma Buma 1976] and [Gallant 1990] and is published in [Schmidt 1991]. This method is tested with the three data sets that have been used before, the Highleyman data (3·55), the different means set (4·28) and the different covariances set (4·29). These tests show that this method is capable of finding a reasonable result with small computational costs.

The central idea of this technique is to train a network by teaching a number of non-linear perceptrons that are adapted successively to the design set. Recall that the non-linear perceptron is optimized by applying a gradient descent on the criterion given by

3·51. Because the inverse function of f() exists, it is also possible to optimize the following criterion function:

$$J_{inv}(\mathbf{w}) = \sum_{y \in D} (f^{-1}(t_y) - \mathbf{w} \cdot \mathbf{y})^2 \qquad (4\cdot30)$$

In this formula $f^{-1}()$ is the inverse function of f(). The advantage of optimizing 4·30 is that the optimal solution can be calculated with formula 3·31. The targets, however, must be replaced by the corresponding inverse values. It should be noted that criterion 3·51 and 4·30 normally yield different solutions. This new criterion has less favorable properties than 3·51. The pattern error function does not approach the minimum error function any more (see section 3·5·2) but is again a quadratic function.

Assume that a network with one *output* unit is trained. The learning is started with a network with only one hidden unit. For this unit the optimal Wiener solution 3·31 is calculated by using the original network targets as the desired response for this unit and by calculating the inverse targets so that 4·30 is optimized. The weights obtained are used as weights $\mathbf{M}_1$ for this unit and these are fixed. The weights of the output unit $\mathbf{W}$ are calculated in a similar way by optimizing the J_{inv} criterion for this unit. In this first pass there are only two weights, namely a threshold and one weight connecting the first hidden unit to this output unit.

A new design set is created by subtracting the current network response from the targets of the original data set. The new targets need to be scaled in a proper range, between 0.1 and 0.9 for example, because the new targets can be negative or larger than one. This scaling is not critical because the weights of the output unit are calculated afterwards, rescaling it to optimize the output unit.

A second unit is added now and the weights $\mathbf{M}_2$ are optimized similar to the first unit, but now using the new target values. These weights are fixed also and the output unit is again optimized, but this time with the outputs of the two units. The design data is evaluated and the current network output is subtracted from the original target values. This process is repeated until all input units are handled.

The algorithm can be summarized in the following steps:

***Step 1**: Use network target values to calculate the Wiener weight vector and set the weights of the first unit ($\mathbf{M}_1$) to these weights. Calculate the weight and threshold of the corresponding output unit, also with the Wiener optimal vector.*
***Step 2**: For each sample: evaluate the current network and subtract the current output from the target values, to determine the new target values. Scale the new target values between 0.1 and 0.9.*
***Step 3**: Add a new hidden unit and set the weights of the new unit $\mathbf{M}_i$ to the Wiener vector for the new target values. Set the output unit to the Wiener vector, using the original target values.*
***Step 4**: If not all the units in the hidden layer are done then go to step 2.*

With this method the first unit is used to approximate the desired signal and the second unit is used to correct for the error made by the first unit. The third unit is used to correct for the error introduced by the first and second unit. At a certain point the optimization can not find a proper vector any more and this will result in units that do not have any correlation with the desired target function. This is a severe problem, specially for the output unit, because the matrix inverse for this unit will become ill conditioned. It is advisable to use Singular Value Decomposition (SVD), see [Press 1988], when solving equation 4·30.

Table 4·2: The results of the non-iterative method for Highleyman data 3·55.

h	Output determinant (inverse of 3·39)	$\hat{\varepsilon}_r$	$\hat{\varepsilon}_t$
1	6.88E-02	0.155	0.1785
2	1.03E-05	0.130	0.1660
3	5.29E-11	0.140	0.1670
4	3.60E-20	0.110	0.1280
5	5.02E-29	0.100	0.1495
6	2.85E-40	0.115	0.1495
7	2.45E-52	0.090	0.1460
8	1.40E-65	0.090	0.1460
9	1.88E-81	0.090	0.1460
10	1.80E-97	0.136	0.1395

Table 4·3: The results of the non-iterative method for data set 4·28.

h	Output determinant (inverse of 3·39)	$\hat{\varepsilon}_r$	$\hat{\varepsilon}_t$
1	8.13E-02	0.175	0.2115
2	2.59E-07	0.170	0.2090
3	7.56E-12	0.170	0.2105
4	1.05E-23	0.165	0.2135
5	6.28E-38	0.165	0.2125
6	3.90E-53	0.165	0.2125
7	1.79E-68	0.165	0.2125
8	1.34E-84	0.165	0.2115
9	5.00E-100	0.165	0.2110
10	2.70E-115	0.160	0.2125

Simulations are performed on a Sun SparcStation1 using ANNLIB (see chapter 8), together with routines from [Press 1988] for the matrix calculations. The inverse matrix in 3·31 needed for the input units must be calculated only once, saving a lot of unnecessary computations. All calculations are done in double precision floating point numbers (53 bits mantissa). The determinant of the inverse matrix in equation 3·31 of

the output unit is calculated, to show that this decreases exponentially when adding more and more units.

The training data of the three distributions used, are shown in figures 3·11, 4·3 and 4·4 and all contain $m=200$ samples. The performances of the obtained classifiers are measured using an independent test set of $m=2000$. The number of hidden units h is varied between 1 and 10.

Table 4·4: The results of the non-iterative method for data set 4·29.

h	Output determinant (inverse of 3·39)	$\hat{\varepsilon}_r$	$\hat{\varepsilon}_t$
1	3.03E-03	0.420	0.4730
2	3.03E-09	0.445	0.4585
3	1.20E-13	0.360	0.4045
4	1.64E-22	0.365	0.4010
5	2.45E-31	0.265	0.2575
6	4.29E-42	0.255	0.2585
7	1.59E-55	0.255	0.2630
8	1.12E-68	0.235	0.2600
9	4.13E-84	0.230	0.2615
10	5.39E-99	0.230	0.2605

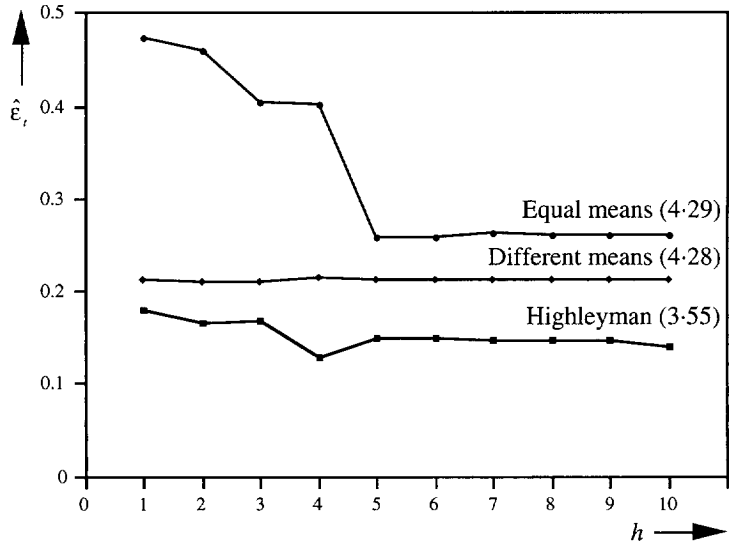

Figure 4·8: The performance measured on an independent test set as function of the number of hidden units h, using the non-iterative training method.

The results of these experiments are shown in table 4·2, 4·3 and 4·4. In figure 4·8 the network performance (measured on the test set) is plotted versus the number of units in

the hidden layer. In figure 4·9 the value of the determinant of inverse matrix in equation 3·31 of the output unit is plotted as function of the number of units in the hidden layer.

For the mean data set (4·28) the optimal classifier is a linear function and figure 4·8 clearly shows that a network with one hidden unit has a good performance and the optimal network uses two units. The determinant of the output unit (see figure 4·9 and table 4·3) rapidly decreases and after 4 units this value decreases with a constant factor of 10^{-15} to 10^{-16}. Since an additional experiment, that was performed with single precision floating point numbers, showed that the slope of the determinant was of the order of 10^{-7}, it can be concluded that the slope of the determinant is based on the machine precision: 53 bits mantissa $\approx 10^{-16}$ and 24 bits mantissa $\approx 10^{-7}$. The last result implies the use of Singular Value Decomposition for solving equation 3·31.

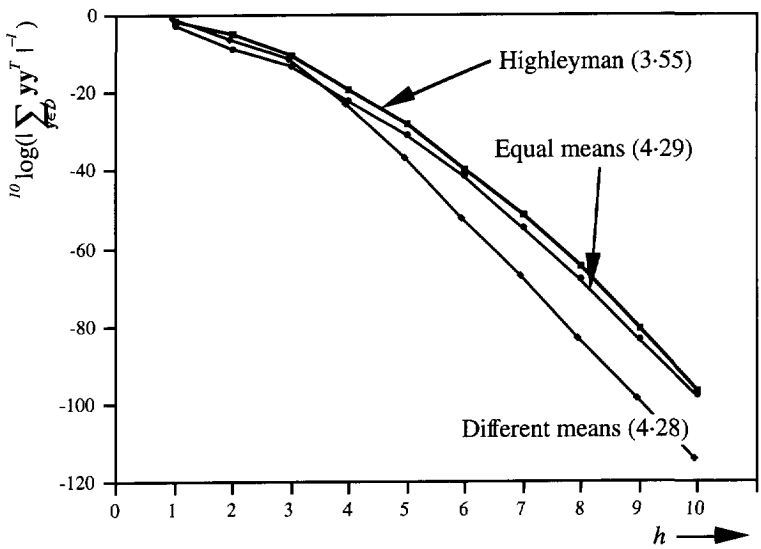

Figure 4·9: The ^{10}log of the determinant of the inverse input correlation matrix, for the output unit as function of the number of hidden units.

The covariances data set (4·29) clearly needs more hidden units to construct the optimal classifier, which is a circle. The optimal classifier is found with 5 hidden units. Although the re-substitution error decreases, it deteriorates on the test set when more than 5 unit are used. We see that the determinant of the output unit decreases with a factor 10^{-16} after 7 hidden units. Note that this is a relatively hard problem to solve, since 100 samples per class in the learning set is not much.

On the Highleyman data set (3·55) the optimal network uses 4 hidden units. The determinant of the output unit decreases with machine precision after 8 units. With 4 hidden units the network approximates the optimal linear decision function. The performance of the optimal classifier is not reached, because probably more learning samples are needed for this method.

In this section a non-iterative method has been described to calculate the weights for a feed forward network with only one hidden layer. The method scales with order d^3 or h^3. That is significantly better than order N^3. Applied to three data sets with known statistical properties and with realistic sample sizes, the algorithm was able to find a set of weights with reasonable performance. A network with 10 hidden units, is calculated in approximately 0.9 seconds on a Sun SparcStation1.

The performance of the algorithm on data sets 4·28 and 4·29 is good, compared to the complexity of the optimal decision function and the number of available learning samples. The performance on the Highleyman data approaches the ideal linear discrimination function but is not able to go beyond that error. However, the Highleyman data set is not a simple problem due to the large difference in variances of the distributions and the limited number of learning samples.

To find the optimal network size it is best to measure the network performance with an independent data set. However, if such a data set is not available the determinant of the output covariance can be used to give an idea of the relative importance of the last added unit. The process of adding units can definitely be stopped if the determinant decreases with a factor that is comparable to the machine precision. In figure 4·9 and tables 4·2 to 4·4 it can be seen that for the three data sets used here this occurs before $h=10$. Therefore this method is omitted in the comparison of section 4·3·4.

The advantage of this method is the computation speed and fast convergence to a reasonable solution. This major disadvantage is related to hierarchical classifiers, the *best two* hidden units are not the same as the *two best* hidden units. If two units are optimized simultaneously, a different solution is expected than if two units are sequentially optimized. A similar problem arises in feature selection that was noticed by [Cover 1974]. Hierarchical classifiers have proved to be valuable in pattern recognition, so it can be expected that the proposed method will be valuable for those situations.

4·4 Interpretation of network training

4·4·1 A simple weight space interpretation

The perceptron training (figure 3·5) and committee machine learning (figure 3·18) can be interpreted in the dual space of the pattern space, namely the weight space. The convergence properties and perceptron theorem are clearly illustrated in this space. For the linear and non-linear perceptron the weight space interpretation does not contribute to more understanding of the training, but illustrates the capacity of these classifiers.

The multi-layer perceptron training is even more complex and in chapter 5 the training will be observed in weight space. The committee machine is contained in the multi-layer perceptron, because the non-linear perceptron approaches the threshold logic unit for large weights. When the weights of the multi-layers perceptron $\mathbf{M}_i$ are

large, the hidden layer of this architecture becomes equal to the input layer of the committee machine (figure 3·17). The weights $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$, in figure 4·1, of the output unit must be set to the value 2 and the threshold $\mathbf{w}_4$ must be set to -3. For this choice of weights, the feed forward classifier becomes equal to the committee machine.

The value of the threshold $\mathbf{w}_4$ can be set to different values, that will result in different voting machines than a simple majority vote. For a machine with three hidden units, three different voting machines are possible. Besides the majority vote, either all units should respond positive (unanimity) or just one (veto) is enough. When the voting strategy is changed the solution space of the hidden units in weight space changes too. The different solution regions, as shown before in figure 3·18 are presented in figure 4·10.

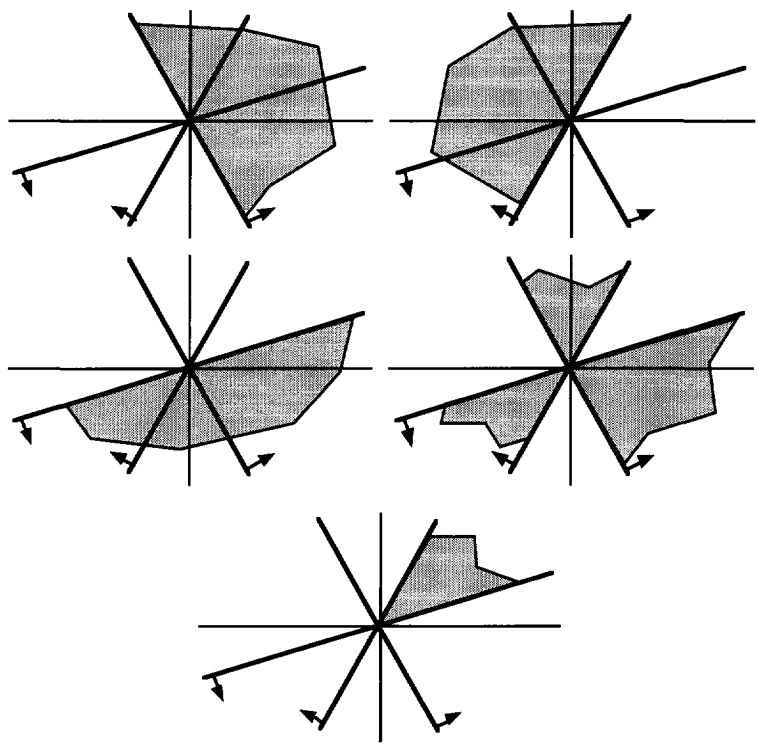

Figure 4·10: A two-dimensional weight space with three pattern hyper planes together with the solution regions for the hidden units (see also figure 3·18). The five different configurations shown correspond to three different voting machines that can all be implemented with a feed forward classifier.

Another generalization is to allow different values for $\mathbf{w}_1..\mathbf{w}_h$, which means that different voting strengths are used. This is already mentioned by [Nilsson 1965, page 98] to increase the classification power, but no general training procedure was known. It is interesting that all these variations are contained in the multi-layer perceptron

architecture and that the training algorithm, like back propagation, can be adapted to a particular choice. This can be achieved by setting the output weights **w** such that the desired voting machine is implemented by the output unit. The training algorithm must ignore any change to these output weights and errors are propagated as before. This demonstrates furthermore the increased classification power of this type of classifier.

4·4·2 Asymptotic properties of multi-layer perceptrons

In section 3·4·4 the relationship between the Bayes decision theory (2·3) and the linear perceptron was shown. The same relation can be proved for the multi-layer perceptrons. It is assumed that when a network is used for a c class classification problem, a network with c outputs is used. The inputs of the network are the d features of a pattern, so the network architecture considered here is of the form d-h_1.. h_i.. h_j-c.

The proof for this classifier follows closely the steps in section 3·4·4 so the intermediate steps are omitted here. The only difference is that not one output exists but c outputs. The sample estimate of the sum of squared errors is replaced by the expected or mean squared error function:

$$J_{mse}(\mathbf{w}) = E[\sum_{i=1}^{c}(\mathbf{t}_{\mathbf{y},i} - g_i(\mathbf{y}))^2] = \int\{\sum_{j=1}^{c}[\sum_{i=1}^{c}(\mathbf{t}_{\mathbf{y},i} - g_i(\mathbf{y}))^2]p(\mathbf{y},\omega_j)\}d\mathbf{y} \tag{4·31}$$

Substituting $p(\mathbf{y},\omega_i) = P(\omega_i \mid \mathbf{y})p(\mathbf{y})$ and exploiting that $g(\mathbf{y})$ is independent of ω_i, the mean squared error can be written in the final desired form:

$$\begin{aligned} J_{mse}(\mathbf{w}) = E[\sum_{i=1}^{c}(g_i(\mathbf{y}) - \sum_{j=1}^{c}\mathbf{t}_{\mathbf{y},i}P(\omega_j \mid \mathbf{y}))^2] + \\ E[\sum_{i=1}^{c}\sum_{j=1}^{c}\mathbf{t}_{\mathbf{y},i}^2 P(\omega_j \mid \mathbf{y})] - E^2[\sum_{i=1}^{c}\sum_{j=1}^{c}\mathbf{t}_{\mathbf{y},i}P(\omega_j \mid \mathbf{y})] \end{aligned} \tag{4·32}$$

The second and third terms in 4·32 are independent of the network parameters, so the optimization of this expression is equal to the optimization of the first term. A possible target coding is:

$$\begin{aligned} \mathbf{t}_{\mathbf{y},i} &= 1 \quad \mathbf{y} \in \omega_i \\ \mathbf{t}_{\mathbf{y},j} &= 0 \quad \mathbf{y} \notin \omega_i \end{aligned} \tag{4·33}$$

The mean squared error procedure thus optimizes the feed forward classifier such that a network output asymptotically approximates, in the mean squared error sense, the *a posteriori* probability:

$$g_i(\mathbf{y}) \approx P(\omega_i \mid \mathbf{y}) \tag{4·34}$$

If an unknown pattern is presented to the network and assigned to the class for which the corresponding output g_i is maximum, this classification rule approaches the Bayes minimum error classifier (equation 2·13). This understanding reveals the relation to

statistical pattern recognition and allows the application of Bayesian theory as presented in section 2·3.

One of the possibilities is to compensate for varying *a priori* class probabilities $P(\omega_i)$. This can for example occur in a medical diagnosis where a significant difference can exist between the class probabilities in the design set and the *a priori* probabilities for the true population. The Bayes formula 2·8 can be used to correct for this imbalance. Besides differences in *a priori* probabilities, equation 2·4 can be used if not a minimum *error* classifier but a minimum *loss* classifier is desired.

The performance of a classifier is normally assessed by measuring the classification error and the final sum of squared errors. The above analysis suggests two other possible figures of merit. A pattern is assumed to belong to a single class, so the network outputs should sum to one. Furthermore if 4·34 is accurate, then the expected value of each output g_i should be the *a priori* class probability ([Wan 1990]). These expected values can be estimated by averaging the outputs over all training examples and measuring the probabilities directly from the training set. The difference can be measured using a probabilistic distance measure, for example the relative entropy suggested by [Richard 1991, page 480].

The Bayesian approximation has been published a number of times in the literature. In [Wan 1990] and [Ruck 1990] different proofs are shown, together with some results for different criterion functions J(). In [Lee 1991a] a similar proof is given, together with an application to handwritten character recognition. The approximation proof for three different criterion functions, together with low dimensional simulation results can be found in [Richard 1991]. A more general theory is presented in [Hampshire 1990], giving the requirements for a criterion function J() to ensure Bayesian approximation. In this paper criterion functions are also proposed that do not approach *a posteriori* probabilities, but where the network output reflects the ranking of these probabilities. On some problems these special criterion functions show better small sample size properties.

A network that uses a squashing function as non-linear function (3·49 for example), can only reach the targets of 4·33 at infinity. When a network is trained using floating point arithmetic, this causes numerical problems. Therefore the following target coding, suggested already by [Rumelhart 1986] is often used instead:

$$\begin{aligned} \mathbf{t}_{\mathbf{y},i} &= 0.9 \quad \mathbf{y} \in \omega_i \\ \mathbf{t}_{\mathbf{y},j} &= 0.1 \quad \mathbf{y} \notin \omega_i \end{aligned} \tag{4·35}$$

The network outputs now approach:

$$(g_1(\mathbf{y}),..g_i(\mathbf{y}),..g_c(\mathbf{y}))^T \approx \begin{pmatrix} 0.9 & 0.1 & 0.1 \\ 0.1 & 0.9 & 0.1 \\ 0.1 & 0.1 & 0.9 \end{pmatrix} (P(\omega_1 \mid \mathbf{y}),..P(\omega_i \mid \mathbf{y}),..P(\omega_c \mid \mathbf{y}))^T \tag{4·36}$$

The matrix in 4·36 is a c by c matrix with value 0.9 on the diagonal and 0.1 for all other elements. The *a posteriori* probabilities can be resolved by solving this set of

equations, by standard numerical techniques such as LU-decomposition ([Press 1988]). Note that the inverse matrix or LU decomposition only needs to be calculated once, when the number of classes is known.

Finally we should not forget that in practice the sum of squared errors criterion 4·7 is an estimate of the mean squared error 4·31 (ignoring a scaling with the number of samples). The optimality of the solution is thus dependent on the sample size and the optimization method. Furthermore the complexity of the chosen network should be sufficient, otherwise a bias towards a higher classification error can result. If the network is, however, too complex, compared to the training sample size, this can limit the true classification error as well.

4·4·3 The optimized internal representation

Discriminant analysis is a general method, based on Euclidean distance, to project data on a subspace by optimizing some criterion. The aim of discriminant analysis is to find a linear subspace, such that the projection of the data onto that space is grouped into well-separated clusters for each class. The Fisher linear discriminant is the most popular and the following criterion is maximized by this function:

$$J_F(\mathbf{w}) = \frac{|\mathbf{w}^T \mathbf{S}_b \mathbf{w}|}{|\mathbf{w}^T \mathbf{S}_w \mathbf{w}|} \tag{4·37}$$

The matrix $\mathbf{S}_b$ is called the *between-class scatter* and is defined as:

$$\mathbf{S}_b = \sum_{i=1}^{c} m_{\omega_i} (\overline{\mathbf{x}}_{\omega_i} - \overline{\mathbf{x}})(\overline{\mathbf{x}}_{\omega_i} - \overline{\mathbf{x}})^T \tag{4·38}$$

Here m_{ω_i} is the number of samples belonging to class ω_i. The *within-class scatter* $\mathbf{S}_w$ is given by:

$$\mathbf{S}_w = \sum_{i=1}^{c} \sum_{\mathbf{x} \in \omega_i} (\mathbf{x} - \overline{\mathbf{x}}_{\omega_i})(\mathbf{x} - \overline{\mathbf{x}}_{\omega_i})^T \tag{4·39}$$

The between and within class scatters are related as follows:

$$\mathbf{S}_t = \mathbf{S}_b + \mathbf{S}_w \tag{4·40}$$

Where $\mathbf{S}_t$, called the *total scatter*, is given by:

$$\mathbf{S}_t = \sum_{\mathbf{x}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T \tag{4·41}$$

The Fisher discriminant searches feature space for the direction for which the between-scatter is maximized relative to the within-scatter. Different criterion functions such as 4·37 are possible. These are all variations of 4·37 (see for example [Devijver 1982, page 247]). The following function has as advantage that it is numerically more stable to compute, especially when the ratio of sample size to pattern dimension is small:

$$J_{num} = \frac{tr(\mathbf{S}_b)}{tr(\mathbf{S}_t)} \tag{4·42}$$

It has been known for a long time in statistical pattern recognition, that the solution of the linear perceptron is equal to the Fisher's linear discriminant, if the target coding of 3·46 is used (see [Duda 1972, page 153]). For a feed forward network with no squashing function in the output unit (4·3), the last part of the network thus performs a discriminant analysis with the hidden unit activation's (4·5) using Fisher's criterion 4·37.

This is mentioned in [Gallinari 1991], [Webb 1990] and [Lowe 1991] where the proof of this relation is republished. Now that the functionality of the output layer is known, the question remains whether the hidden layer transformation (equation 4·5) optimizes some kind of scatter criterion too.

In [Gallinari 1991] various networks with different numbers of hidden *layers* are trained and the scatter $tr(\mathbf{S}_b)/tr(\mathbf{S}_t)$ is calculated for successive hidden layers' activations. For their example the scatter gradually increases from input layer to output layer. The interpretation of this result should be taken with care because a different number of units per layer were used. Values of the scatter $tr(\mathbf{S}_b)/tr(\mathbf{S}_t)$ for different dimensions are difficult to compare, but at least their experiment hints that the hidden layer is performing some kind of feature extraction.

The network is optimized as a complete system, using the sum of squared errors criterion 4·7. This means that the input to hidden units transformation is coupled to the hidden to output transformation. This observation leads [Webb 1990] to the following conclusion:

> *Excerpt from [Webb 1990, page 369]:* Choosing optimum weights to minimize the square error at the output of the adaptive layered network (**W**), force the first set of weights (**M**) to be chosen such that the transformation from the input data to the output of the final hidden layer, maximizes a Fisher criterion in the space spanned by the outputs of the final hidden layer (see 4·5). The network is constructed in such a way that the final layer performs an optimum separation of patterns into their classes by a linear transformation. To minimize the error of this procedure, it is necessary for the first part of the network to perform an appropriate non-linear transformation of the original patterns into a space in which discrimination is made easier. This non-linear transformation has to be such that the overall network discrimination is maximized. The space generated by the outputs of the hidden units is the crucial one in which pattern analysis and recognition should be performed. The final linear classifier used in the network could be replaced by a more sophisticated statistical pattern analysis method that would benefit from the discriminatory power of the feed forward network.

This statement, in my opinion better formulated as a hypothesis, can be analyzed in depth by looking in detail at what happens during training. This analysis is focused on a network with one hidden layer and one output unit (*d-h-1*), as shown for example in figure 4·1. It is furthermore assumed that the design set only contains two classes ω_1 and ω_2.

The original features **y** are mapped into a new vector **y'** by a non-linear transformation given by equation 4·5. Here **y** is a $d+1$ dimensional vector and **y'** is a

$h+1$ dimensional vector. Both vectors are augmented with a value 1. The space of the first h features of $\mathbf{y}'$ is restricted to the open subspace $<0,1>^h$, due to the squashing functions. The output unit performs a linear discriminant ($\mathbf{w}$) in this h dimensional space.

Assume that a random sample ξ is chosen from the design set D and that all the weights are updated according to the back propagation rule 4·13. As a result of this update, all patterns $\mathbf{y}'$ will now move in the subspace to a new position. This move $\Delta\mathbf{y}$ is dependent on the change of the weights contained in $\mathbf{M}$. If the generalized delta rule 4·13, 4·11 and 4·12 are combined, the following change in $\mathbf{M}$ can be derived:

$$\begin{aligned} \Delta\mathbf{M}_i &= \eta f'(\mathbf{M}_i\xi) f'(\mathbf{w}\cdot\xi')\mathbf{w}_i(1-g(\xi))\xi & \xi\in\omega_1 \\ \Delta\mathbf{M}_i &= -\eta f'(\mathbf{M}_i\xi) f'(\mathbf{w}\cdot\xi')\mathbf{w}_i g(\xi)\xi & \xi\in\omega_2 \end{aligned} \tag{4·43}$$

This result can now be used to calculate the changes $\Delta\mathbf{y}$ of the individual components of the samples $\mathbf{y}'$ in hidden space:

$$\Delta\mathbf{y}_i = f(\mathbf{M}_i\mathbf{y}+\Delta\mathbf{M}_i\mathbf{y}) - f(\mathbf{M}_i\mathbf{y}) \approx f'(\mathbf{M}_i\mathbf{y})\Delta\mathbf{M}_i\mathbf{y} \tag{4·44}$$

This last term, a first order Taylor approximation, is permitted if the learning rate η is chosen small enough. Combining 4·43 and 4·44 results in:

$$\begin{aligned} \Delta\mathbf{y}_i &= f'(\mathbf{M}_i\mathbf{y})\eta f'(\mathbf{M}_i\xi) f'(\mathbf{w}\cdot\xi')\mathbf{w}_i(1-g(\xi))\xi\cdot\mathbf{y} & \xi\in\omega_1 \\ \Delta\mathbf{y}_i &= -f'(\mathbf{M}_i\mathbf{y})\eta f'(\mathbf{M}_i\xi) f'(\mathbf{w}\cdot\xi')\mathbf{w}_i g(\xi)\xi\cdot\mathbf{y} & \xi\in\omega_2 \end{aligned} \tag{4·45}$$

To obtain the expected update of a component $\mathbf{y}_i$, the expectation for the design set must be calculated. This results in the final equation:

$$\begin{aligned} E[\Delta\mathbf{y}_i] = \eta f'(\mathbf{M}_i\mathbf{y})\mathbf{w}_i\{&P(\omega_1)E_{\xi\in\omega_1}[f'(\mathbf{M}_i\xi) f'(\mathbf{w}\cdot\xi')(1-g(\xi))\xi\cdot\mathbf{y}] \\ &-P(\omega_2)E_{\xi\in\omega_2}[f'(\mathbf{M}_i\xi) f'(\mathbf{w}\cdot\xi')g(\xi)\xi\cdot\mathbf{y}]\} \end{aligned} \tag{4·46}$$

No direct conclusions can be derived from this equation unless the following property holds for the design set:

$$\begin{aligned} \xi\cdot\mathbf{y} > 0 \qquad & \forall\xi,\mathbf{y}\in\omega_1 \text{ and } \forall\xi,\mathbf{y}\in\omega_2 \\ \xi\cdot\mathbf{y} < 0 \qquad & \forall\mathbf{y}\in\omega_1,\xi\in\omega_2 \text{ and } \forall\mathbf{y}\in\omega_2,\xi\in\omega_1 \end{aligned} \tag{4·47}$$

In equation 4·46 f'() is always positive because a squashing function is a monotonic increasing function. Furthermore g($\mathbf{y}$) and (1-g($\mathbf{y}$)) are always positive too. Using property 4·47, the expected value of $\Delta\mathbf{y}'$ is a positive fraction of $\mathbf{w}$, for patterns from class ω_1 and a negative fraction of $\mathbf{w}$ for all patterns belonging to ω_2. For this special case it can be concluded that *any* update will cause the samples of different classes to be moved in the opposite direction in hidden space representation. This conclusion is unfortunately only true for those data sets for which property 4·47 holds. Such a data set is also linearly separable and it is probably better to use a simple threshold logic unit in this case.

From equation 4·46 it can be concluded further that the expected update is normally determined by a subtle balance between the expected values in this equation. This

balance is dependent on **M** and **w** and because these weights change during the training, the effective change in this internal space can reverse at some point. Any conclusion about the optimized internal representation is thus dependent on the number of updates used.

In [Webb 1990] the authors considered networks with no squashing function in the output unit (equation 4·3). For this particular case formula 4·46 reduces to:

$$E[\Delta \mathbf{y}_i] = \eta f'(\mathbf{M}_i\mathbf{y})\mathbf{w}_i\{P(\omega_1)E_{\xi\in\omega_i}[f'(\mathbf{M}_i\xi)(1-g(\xi))\xi\cdot\mathbf{y}] - P(\omega_2)E_{\xi\in\omega_2}[f'(\mathbf{M}_i\xi)g(\xi)\xi\cdot\mathbf{y}]\} \quad (4\cdot48)$$

There are no restrictions for the weights **w** connecting the hidden units to the output unit, so in this case g(**y**) and (1-g(**y**)) are not bounded any more. Condition 4·47 is not sufficient any more to make any prediction about what will happen in hidden space when the network weights are updated. It has even become more unlikely that some kind of scatter is optimized by the hidden units because output units change rapidly during training.

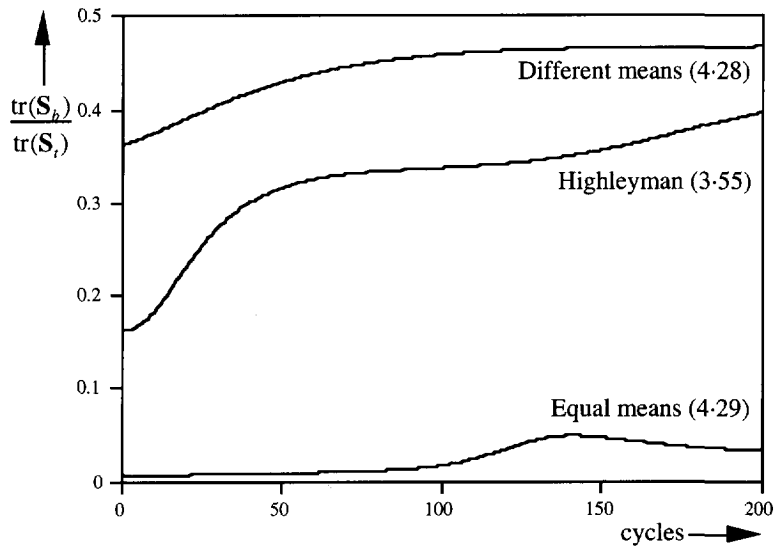

Figure 4·11: Internal scatter as a function of the number of training cycles, measured for three data sets. The output unit is a non-linear perceptron.

A number of experiments have been performed to obtain more understanding of the dynamics of equation 4·46 and 4·48. This formula can be directly evaluated for different data sets, but the original hypothesis concerned the optimizing of some scatter related criterion. Therefore the scatter function $\mathrm{tr}(\mathbf{S}_b)/\mathrm{tr}(\mathbf{S}_t)$ is measured during the training of a 2-8-1 network. The three data sets that have been used so far, shown in figures 3·11, 4·3 and 4·4, are used again. The generalized delta rule (BP) with η=0.01

and α=0 is used for data both sets 3·11 and 4·3 and parameters η=0.05 and α=0.5 is used for data set 4·4.

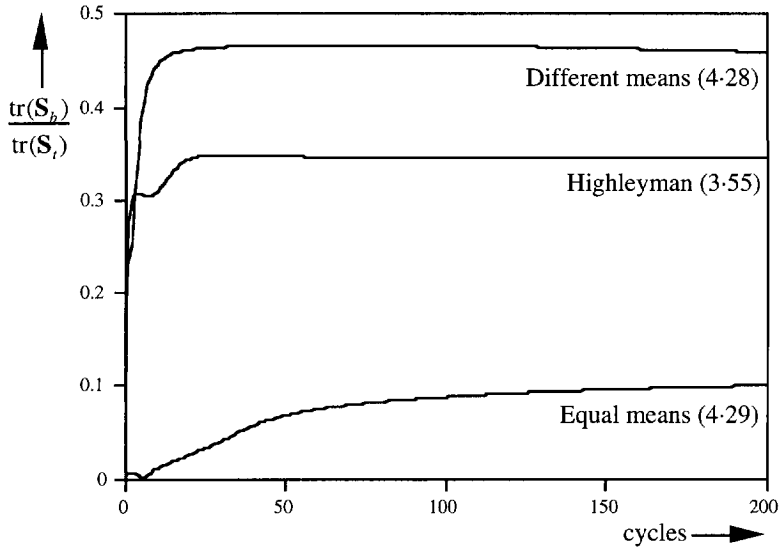

Figure 4·12: Internal scatter as a function of the number of training cycles, measured for three data sets. The output unit is a fixed *non-linear perceptron.*

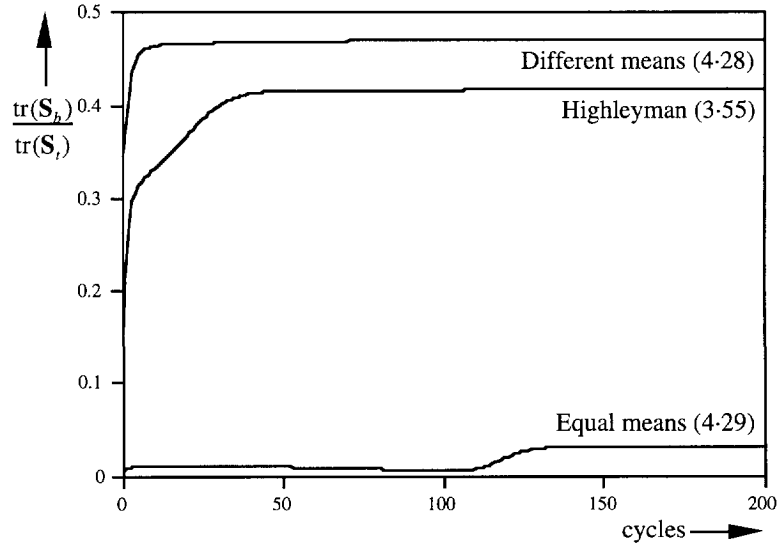

Figure 4·13: Internal scatter as a function of the number of training cycles, measured for three data sets. The output unit is a linear *perceptron.*

The scatter at the hidden units is measured as a function of the number of cycles that the complete training set has passed. Two different types of networks are trained, a network with a squashing function at the output unit (4·2) and one without a squashing function at the output (4·3). The dynamics of these networks are both dependent on the weights at the input layer (**M**) and the weights at the output unit (**w**). To analyse the influence of both *coupled* processes, the experiments are also performed with networks with fixed, preset weights at the output unit. The weights are set such that the output unit is the diagonal hyper-plane of the subspace $< 0,1 >^h$ passing through the origin.

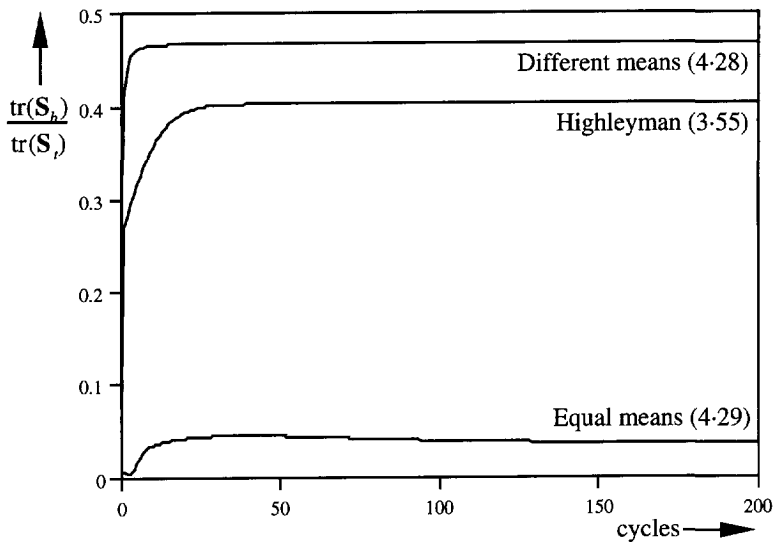

Figure 4·14: Internal scatter as a function of the number of training cycles, measured for three data sets. The output unit is a fixed *linear perceptron.*

The results of the simulations for networks with non-linear output are shown in figure 4·11 and 4·12 and for the networks with linear output the results are plotted in figures 4·13 and 4·14. Indeed these experiments reveal that the hidden units optimize a scatter related criterion. The networks for which the output units are fixed to preset values show a more rapid convergence to a value that is roughly also attained by networks that are completely trained. Networks with no squashing function at the output show significant differences during the training but seem to approach the same asymptotic values. In figure 4·11 data set 4·29 shows a decrease after 140 cycles. This data set is a typical example of a data distribution that is highly unsuitable when optimizing a scatter like criterion. This decrease shows that indeed the expected update is determined by a subtle balance between the two expectations that are found in equations 4·46 and 4·48.

4·5 Recent developments

The attention that the multi-layer perceptron has received, resulted in an enormous volume of research and publications. An excellent and exhaustive review of recent research, including a reference list of more than 250 publications, can be found in [Xu 1992]. In this section two topics, that are interesting for pattern recognition and training networks, will briefly be discussed.

4·5·1 An alternative network model

One alternative network model shows a close relation with the Parzen window classifier [Duda 1973, page 88]. The mathematical formulation of this type of network is given by:

$$g_i(\mathbf{y}) = \sum_{j=1}^{h} \mathbf{W}_{i,j}\, f(|\,\mathbf{y} - \mathbf{M}_j\,|) + \mathbf{W}_{i,h+1} \tag{4·49}$$

The non-linear function f() is in this case *not* a squashing function, but the type of function that is also used as a Parzen kernel. The most common one is the Gaussian function:

$$f(x) = e^{-x^2/s^2} \tag{4·50}$$

This type of network, called *radial basis function network*, has been shown to be optimal for function approximation. The related theory and properties are discussed in [Poggio 1989]. The parameters $\mathbf{W}$, $\mathbf{M}$ and s, that may differ per unit, can be trained by one of the methods that are discussed in this chapter. In [Hartman 1990] this type of network is shown to have universal approximation capabilities, similar to the multi-layer perceptrons found so far.

The number of kernels (units) is chosen to be equal to the number of design patterns in the Parzen density estimation. The weights $\mathbf{W}$ are chosen equally and set to a fixed normalization parameter. The weight vectors $\mathbf{M}_i$ are the patterns found in the design set. For the network described by formula 4·49, the number of hidden units is normally less than the number of design patterns and the individual weighting of the units is different. If the network outputs are optimized using criterion 4·31, radial basis networks will approximate *a posteriori* Bayes probabilities. These networks can therefore be characterized as reduced kernel Parzen classifiers.

4·5·2 Random search training techniques

The optimization techniques of sections 4·3·1 and 4·3·3 that were used to train a network, were all based on derivative information. A search direction or step was constructed by calculating the gradient or higher order derivatives. Techniques that are called *random search methods*, perturb each weight and observe whether the change in criterion, ΔJ is acceptable or not. Two types of algorithms can be distinguished.

The first technique is described by [Patrikar 1990] and only one weight is changed at a time. At random, one parameter is selected and the value is changed by a positive variation. If ΔJ is negative this change is accepted, otherwise a negative variation is applied and accepted if ΔJ is negative. If no variation of the selected parameter reduces the error J than the weight is restored to its old value. At random the next parameter is chosen and this process is repeated until some convergence criterion is reached.

The second type of random search methods, perturb all weights collectively at each step instead of just one weight at the time. A search vector is chosen at random and a positive fraction is applied to the network. This change is accepted if this reduces the criterion J, otherwise a negative fraction is tried, similar to the algorithm described above. A new search vector is chosen at random and this process is repeated until the error is sufficiently reduced. The condition of accepting a change can be relaxed by using the *simulated annealing* [Kirkpatrick 1983] approach for escaping local minima. The acceptance of a perturbation is now given by the probability:

$$p = \begin{cases} 1 & \Delta J \leq 0 \\ e^{-\Delta J/\eta} & \Delta J > 0 \end{cases} \tag{4·51}$$

The parameter η controls the training, it starts from an initial value and is gradually reduced to zero during the training process. This algorithm is applied to feed forward networks in [Day 1990].

4·6 Discussion

In this chapter the *basic* theory of multi-layer perceptrons has been discussed. Clearly the simple perceptrons are theoretically much better established. A thorough understanding of these simple structures gives some hints and understanding of the more complex structures.

Table 4·5: Properties of multi-layer perceptrons.

Type	Data requirements	Decision regions	Convergence properties	Asymptotic properties
Multi-layer perceptron		Convex and non-convex regions of general shape	Local SSE convergence	Approximately unbiased

The limitations of the simple perceptrons do not apply anymore to the multi-layer perceptrons, with the expense that less understanding exists of what is going on. The feed forward classifier is more or less a black box that after training is performing a type of non-linear discriminant analysis, followed by a linear classification. The network outputs can be interpreted as Bayes probabilities and this knowledge

establishes the link to statistical pattern recognition. In table 4·5 the properties of feed-forward classifier are listed. Additionally it should be noted that the listed properties are dependent on the network complexity. It is assumed that the sufficient number of hidden units h is known *a priori*.

Finally a number of different training methods have been compared for three pattern recognition tasks that differ in covariance structure. It is surprising that the generalized delta rule scores relatively well, because it is the most simple technique. The conjugate gradient descent method, however, does not need any user interaction to find a set of stable training parameters and is therefore favorable.

5
Training a Network in Practice

5·1 Introduction

Forward classifiers have become one of the most widely used classification techniques in recent years. The training algorithm, the generalized delta rule, is simple and elegant from a mathematical and implementational point of view. The training, although often *very* slow in practice, is easily mapped to parallel machines if the processing power of sequential machines is not enough. In recall mode, if the network is trained, the feed forward classifier is generally fast and special hardware exists to perform real-time classification if needed. A chip set is available from Intel for example (see [Holler 1992]), which implements a general feed forward network and the user can download the weights.

In the literature a number of impressive classification problems have been solved by using *back propagation* trained networks. The NETtalk project from Sejnowski and Rosenberg ([Sejnowski 1986]) is well known, where a network is trained to predict the relationship between written English text and the corresponding phonetic representation. In [Pomerleau 1989] a feed forward network is trained to navigate a car on the road using range and image information. A feed forward network became the 1989 winner in computer backgammon playing ([Tesauro 1990]). In [Le Cun 1989] these networks have been *successfully* applied to the problem of handwritten ZIP code recognition.

In *some* cases the neural network implementation shows equal performance with the best known classical method whereas the solution is reached must faster without *long research* ([Sejnowski 1986]). In most cases the neural network approach is just a black box to the user. Once a training set is constructed the algorithm will solve the classification task. In chapter 4 it has been shown that the class of feed forward networks can approximate any continuous function given enough hidden units. With the selection of an appropriate number of hidden units, this method can solve, in theory, any classification task wanted.

Despite the enthusiasm for this method, the user will notice that in practice it is not at all easy to solve a particular problem with this method. I agree with [Weiss 1991, page 108] that: *The back propagation training procedure can be a user's nightmare*. First the back propagation can be computationally *extremely* slow, depending on the problem to solve. Furthermore a number of practical problems arise for which only heuristic

solutions are available. These problems vary from problems also found in pattern recognition, such as feature selection, to problems specific to the algorithm. The back propagation method is known, for example, to be sensitive to the initial conditions and the training may be regarded as an unstable system from the traditional systems viewpoint.

A good example of this sensitivity is shown in figure 5·1. In this graph two curves are plotted of the training process in time, the mean square error (see next section) versus the number of training sweeps. In both situations exactly the same network architecture, learning set and training parameters were used. The only difference is the set of initial weights chosen to start the learning procedure. In this figure we see a large difference between the learning curves.

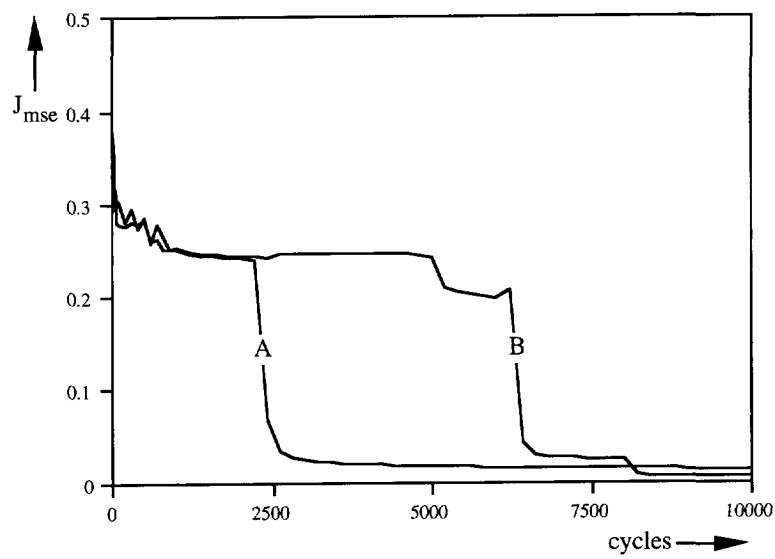

Figure 5·1: The mean squared error J_{mse} (5·1) versus the number of training cycles, for a 2-3-2 network and 16 training samples. Two initializations A and B, with learning parameters η=0.5 and α=0.01 are shown.

The main objective of this chapter is to obtain more understanding of the difficulties that arise in training a feed forward neural classifier. An important aspect is the complexity of the training procedure imposed by the optimized sum of squared errors criterion function. Furthermore, a detailed understanding about the sensitivity of the training method and the consequence of this phenomenon for the classification performance is an important issue in classifier design.

In this chapter I propose that training a neural network using the back propagation method is a stochastic process. A training session is no more than one draw from this random process and, after training, a specific network is obtained that is dependent on a probability distribution describing the learning process. Any change to the algorithm or

parameter setting will change the underlying probability distribution that controls the learning.

The objective function that is optimized to find a set of weights is the squared error criterion on a random learning set. It is obvious to choose the parameter settings or heuristic improvement such that the probability of obtaining a network with a low value of this criterion becomes larger. In pattern recognition, however, the probability of error or, better, the generalization of that for the whole population is the main objective. Any change to the training algorithm, whether or not supported with strong theoretical arguments, should improve this criterion. Important in this context is the joint probability distribution of the optimized criterion and the probability of error for the entire population.

The work presented in this chapter is a result of close cooperation with Sarunas Raudys of the Data Analysis Department, Institute of Mathematics and Informatics, Vilnius, Lithuania. An excerpt of these results is published in [Schmidt 1993b] and the following presentation of this joint research reflects my interpretation of the large number of simulations that were performed.

5·2 Some definitions

In the previous chapter, the training algorithm was formulated as the optimization of the sum of squared errors (J_{sse}). To compare different training sessions on data sets with a different number of training samples, it is easier to compare the mean squared error instead. The mean squared error was introduced earlier in 3·42 and 4·31 but in those formulas the asymptotic definitions were presented. In this chapter the sample estimate of the mean squared error is used and is simply related to the sum of squared errors as follows:

$$J_{mse}(\mathbf{w}) = \frac{J_{sse}(\mathbf{w})}{m} \tag{5·1}$$

I would like to emphasize, however, that this criterion is a function of the *training* samples and is in that sense a re-substitution measure. The performance of a network classifier can also be measured as the number of mis-classified samples. A sample is regarded as mis-classified if the difference between the output $g_i(\mathbf{y})$ and target $\mathbf{t}_{y,i}$ is larger than a certain threshold. In all examples the target coding of 4·35 is used and the threshold is set to 0.4. The performance of a network can be measured on the training set ($\hat{\varepsilon}_r$) or using an independent test set ($\hat{\varepsilon}_t$).

In sections 5·4 and 5·5 some examples will be shown to illustrate the complexity of the error surface and the problems that can arise when training a network for a specific task. In all examples the standard back propagation method, formula 4·13, is applied to a network with one hidden layer and 3 units in that layer (*d-3-1*).

The data sets that are used in these sections contained two different classes. The statistics of the data set are given by:

$$\mathbf{x}_{\omega_1} = \begin{pmatrix} 6.2\cos(\gamma_1)+\zeta_1 \\ 6.2\sin(\gamma_1)+\zeta_2 \end{pmatrix}, \quad \mathbf{x}_{\omega_2} = \begin{pmatrix} 10\cos(\gamma_2)+\zeta_3 \\ 10\sin(\gamma_2)+\zeta_4 \end{pmatrix} \tag{5·2}$$

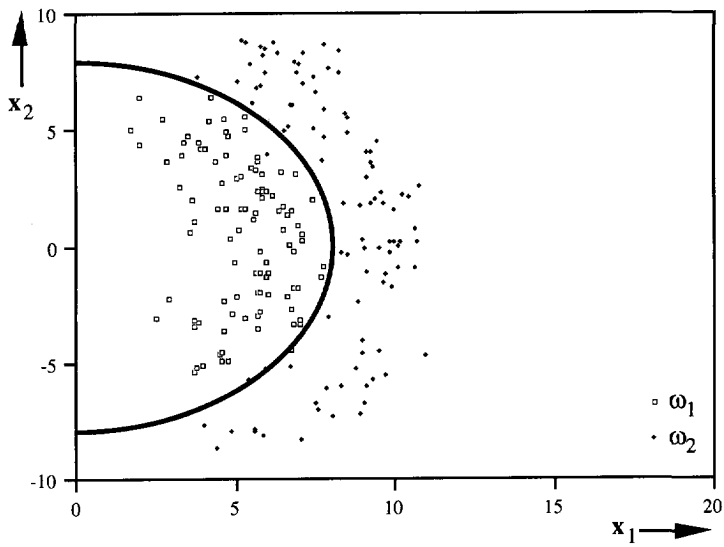

Figure 5·2: An example training set of 200 samples for patterns distributed with statistics given by equation 5·2. The quadratic classifier shown approximates the Bayes optimal classifier.

In this formula ζ_i is a Gaussian random sample with zero mean and unit variance and γ_i is a uniform distributed random sample from the interval $[-\pi/3, \pi/3]$. The Bayes error ε^* is approximately 2.9%. An example training set for this distribution is plotted in figure 5·2.

5·3 Design considerations

When a feed forward classifier, together with the generalized delta rule, is applied to solve a particular problem, the user is confronted with a large number of design considerations. A number of choices are dependent on each other and finding an optimal setting is almost an optimization problem itself. The following list (subsections) is not claimed to be complete but is representative as a practical list. See [Xu 1992] for an exhaustive list of recent developments.

5·3·1 The input representation and number of samples

The inputs presented to the networks, the measurements that will be used by the classification system, are of course important for the classification result. Which measurements to be used and the pre-processing of this data is called feature selection in pattern recognition and all known methods in this field can be applied to neural networks training (see section 2·2·2).

A pre-processing technique is for example the de-correlation of the input signals, as explained in [Orfanidis 1990], on an orthogonal basis defined by the principal directions in the input space. It can be shown that for linear systems the training will be faster if a system is taught with de-correlated and scaled signals. Furthermore it is possible to use only a subset of these de-correlated signals, those with the correspondingly largest eigenvalues. This reduction of input signals can improve the classification performance for specific problems, especially if only a few learning samples are available.

A problem directly related to the question of what inputs should be used by the neural classifier is the question of how many learning samples should be used. The answer is the more the better because the generalization capability of the classifier increases with the number of training patterns used. It is obvious that infinite learning and testing sets exist for artificial data sets only and that in practice a finite set must be used.

A theoretical bound on the number of samples needed to ensure good generalization is given by [Baum 1989] using the framework introduced by Vapnik and Chervonenkis ([Vapnik 1982]). Unfortunately this bound is very high if one requires a network solution not far from the Bayes optimal solution. For practical situations this bound is useless (see also section 6·2).

5·3·2 The output representation

In multi-class classification problems a choice must be made as to how to point to a specific class membership by a feed-forward network. The success of network training, the ease of separability in this case, can depend on how different classes are encoded. The most commonly used method is to define c outputs if c different classes are possible, as already explained in chapter 4.

This coding scheme is one possibility to encode c classes and seems to work very well in practice. The asymptotic properties of the multi-layer perceptron (section 4·4·2) support this coding scheme and it can be concluded that without any further knowledge this is a good choice.

A different coding is used by [Sejnowski 1986] where 52 phonemes are coded with 22 basic voices that have been distinguished by experts. The 52 different classes are represented by prototype vectors in this 22-dimensional space. The prototype with the smallest angle to the network output vector specifies the class. In this classification example the final performance is significantly improved by this *heuristic* method.

Furthermore it is observed that when the network classifier makes an error, it is often close in sound to the correct phoneme.

5·3·3 The reject values

A problem related to the output coding is the ability to reject samples to improve the classification performance. In this case the outputs should reflect *posteriori* probabilities in order to be able to reject samples in a statistically significant way. According to [Hamshire 1990], [Ruck 1990] and [Wan 1990] this is possible if we optimize the mean square error together with a class encoding as mentioned in the previous section. The network outputs can be interpreted as probabilities and samples can be rejected if the output value is below a certain threshold.

5·3·4 The network architecture

The number of hidden units arranged in the hidden layers plays an important role in the complexity of the classifier. Take for example a network with one hidden layer and vary the number of units of that layer from one to many. The network classifier can be expected to move from an under-fitting to an over-fitting situation, which means that in the first case the Bayes performance is not reached because the classifier is not powerful enough. In the second case Bayes performance is not obtained because the classifier is too well adapted to the learning data including its statistical fluctuations. This means that there is a *magic* number of hidden units that will match the network to the complexity and sample size of the data.

There are a number of heuristic rules to choose the number of hidden units. A proposed heuristic is that a network should average at least ten samples per weight (see [Weiss 1991 page 104]). A theoretical approach is suggested by [Amari 1992a] using an information criterion to determine to the optimal number of hidden units. This theory is a statistical approach to determine if an extra hidden unit should be added to the network. See also [Moody 1992] on this topic.

Despite heuristics the probability of error measured on an independent test set, $\hat{\varepsilon}_t$, is far more accurate to estimate the optimal number of hidden units and should be chosen whenever possible. The network architecture in relation to the data complexity and sample size is still a topic for further research (see also chapter 6).

5·3·5 Regularization

The regularization theory introduces a different optimization criterion to improve the classifier or the function approximation. This method assumes some prior knowledge about the smoothness of the solution or some prior knowledge about the optimal weight combination. The following function is an example of regularization where the parameter β controls the amount of regularization.

$$J_{reg}(\mathbf{w}) = J_{sse}(\mathbf{w}) + \beta \mid \mathbf{w} - \mathbf{w}^* \mid^2 \qquad (5\text{·}3)$$

The second term is called an attraction term and will force the solution around some value $\mathbf{w}^*$ dependent on the value of β that determines the attraction *force*. The value of $\mathbf{w}^*$ is of course in practice not known and must be somehow guessed. One choice is $\mathbf{w}^*=\mathbf{0}$ favoring small weights, which implies that the *more* linear solution is preferred (see for example [Krogh 1991]). A new practical problem arises now. How large should the value of β be chosen? The relation between regularization and network optimization without over-training to be sure of not adapting the network too much to the training set is mathematically investigated in [Sjoberg 1991].

A common regularization method is the following:

$$J_{reg}(\mathbf{w}) = J_{sse}(\mathbf{w}) + \beta \sum_{\mathbf{x} \in D} \left| \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} \right|^2 \tag{5·4}$$

The second term in equation 5·4 is the average squared magnitude of the gradient of the output on the training points. Solutions with a small squared magnitude of the gradient are solutions that are smooth. Parameter β balances the tradeoff between smoothing and minimizing the original loss function. Again there is a problem of how to guess a good value for β if no knowledge about the optimal solution exists. This problem is similar to the choice of the optimal smoothing parameter in Parzen window classifiers [Duda 1973, page 88].

5·3·6 The random initial state

The learning procedure starts with a network in a random state; the weights and biases are initialized with some choice of random numbers. The random initialization greatly influences the final network and different networks will be obtained from one initialization to another. The training can be repeated a number of times and the network with the best performance is selected from this set. The number of times the learning procedure must be repeated may differ from one application to the other. In [Kolen 1991] the sensitivity of the back propagation to the initial weights is discussed.

5·3·7 Batch updating or sample updating

In equation 4·10 the gradient is estimated using the entire learning set. The revision of the network weights is thus performed if the complete learning set is passed (called epoch or batch updating). Despite the strong mathematical arguments for this method, revision after each sample is popular and more commonly used. Partial batch updating is also possible. Now the weights are updated if a fraction of the learning set is evaluated. What method is the best depends on the application and has to be chosen by trial and error. In [Sejnowski 1986] for example, one word generates on average 5 samples and this known structure in the data set leads the authors to the choice of updating the weights when a complete word is passed. This is a partial batch updating where the batch size varies during training.

If sample updating is chosen, the sequential order of presentation can be important. The order of presentation can be kept constant during learning or a reshuffling after one epoch can be done. Whatever is the best solution depends on the application, learning rate, etc.

The sample-update method mentioned in 4·13 is the most popular method although it does not follow the true gradient. From an empirical study, supported by some theoretical arguments, [Le Cun 1989] claims that this method is much faster, especially on large redundant data bases.

Stochastic gradient training algorithm changes weights based on a single training pattern. Therefore even in a case of perfect training (when the true J_{sse} achieves its minimum) the gradient of the single pattern J_{sse} will not be zero. If the learning rate η is not zero at this point, this will result in a change in the weights. As a result of the stochastic nature of the data an oscillation around the weights of the minimum may occur. The J_{sse} will now become a random variable with a distribution function. The mean and variance of this distribution will depend on the value of the parameter η as well on the architecture of the classifier, the particular set of the training patterns, and surely on the number of the training sweeps (how far we are from the local or global minimum).

Not far from the minimum point the gradient will be small, consequently the oscillation mentioned will be tend to be small too. These oscillations play a positive role since they help to find solutions that are more robust, according to [Le Cun 1989].

5·3·8 The learning rate and momentum term

The learning parameters η and α are of critical importance in finding the true global minimum. A very small learning rate η will make the learning very slow and will end more likely in the nearest local minimum. A too large learning rate results in an oscillation in learning between relatively poor solutions. For the linear system (perceptrons) the optimal learning rate can be determined from the input correlation matrix (see section 3·4·3). This does not hold, however, for non-linear systems as described by equation 4·10.

If the sample-update method is used the learning will never really converge but at the end will *jiggle* around the true minimum because the sample updating only approximates the true gradient. To let the learning converge to the real minimum the learning rate should be reduced with the number of training sweeps to enable convergence to one solution. The proper way to decrease the learning rate with the number of training sweeps is analyzed by [Lou 1991] and can be applied to the non-linear networks as specified by equations 4·2 and 4·3.

5·3·9 Adding noise during training

A far less detectable problem is the convergence to a local minimum. When this happens the training has all the appearance of convergence although a better solution in

mean square error exists. This problem can be avoided with one of the following methods.

The first method adds some amount of randomness (noise) to the samples during learning. This will result in some small error in the measurement of the gradient that gives the possibility of escaping from a local minimum. This method is also suitable for classification problems where only a few learning samples are available and the danger exists that the network will adapt too much to the learning data. By adding noise to the learning set the data is artificially increased.

In [Reed 1992] the theoretical relationship is shown between regularization and using jittered training data. Assume that identical independent distributed (iid) noise samples with zero mean and covariance $\sigma^2\mathbf{I}$ ($\mathbf{I}$ is the unit matrix) are added to the training samples. It can be shown that the first order approximation of the loss function optimized in this case can be written as:

$$J(\mathbf{w}) \approx J_{sse}(\mathbf{w}) + \sigma^2 \sum_{\mathbf{x} \in D} | \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} |^2 \qquad (5\cdot5)$$

If equations 5·5 and 5·4 are compared, it is obvious that these equations are equal to the first order approximation with $\beta = \sigma^2$. In this case the variance of the noise plays a similar role as the regularization parameter β. Again the same problem occurs as in regularization, that *a priori* information is needed about the smoothness of the solution to know how the choose to the amount of noise σ^2.

A second possibility to avoid local minima is to add a small amount of randomness to the weights during the learning and to reduce this amount with the number of sweeps. This is a combination of gradient descent and random search. Again this method introduces the problem of how fast should the noise term must be reduced during the learning ([Lehman 1988]).

5·3·10 The stopping criterion

The iterative procedures of equations 4·10 or 4·13 do not specify a condition when to stop. The question that arises now: what is a good stopping criterion? If an independent data set is present the iteration can be stopped just before the performance (probability of error) increases on this data set. An extra data set can be created by dividing the learning set into two parts and using one part to train the network and the second part to stop the iteration. If an unbiased estimate of the expected performance of the trained network is needed then a third independent data set is needed. There are of course situations where it is too expensive to obtain three data sets (which are large enough) or when there is simply not enough data present.

The most obvious heuristic is to just limit the number of training epochs to a fixed number and stop if this is reached. In [Sjöberg 1991] the authors show that there is a strong theoretical relationship between the regularization method described in equation 5·3 and the use of a limited number of training epochs.

Another possibility is to stop if the the mean square error did not decrease more than a certain threshold during the last epochs. A combination of these methods is possible.

An alternative, less heuristic method, applies a method from the statistical pattern recognition literature, the editing algorithm (see [Devijver 1982]), to remove the overlap between the classes in the learning set. This editing results in a performance that is close to Bayes optimal and from the resulting learning set, it is known that the training should continue until all samples are correctly classified. A full explanation of this method can be found in [Kraaijveld 1990].

5·4 The complexity of the error surface

In section 3·5·3 an example error surface was shown for the linear and non-linear perceptrons (figure 3·12). The error surface of the linear perceptron is convex shaped and has only one minimum. For the non-linear perceptron the error space is generally not convex and in section 3·5·1 it was noted that it is possible that criterion 5·1 contains local minima. In figure 5·3 this is illustrated for a non-linear perceptron with $m=32$ samples distributed according to the statistics given by equation 5·2.

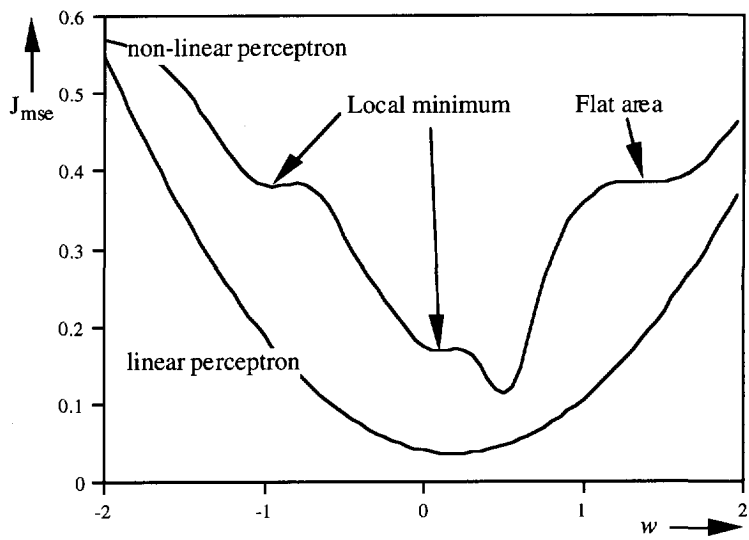

Figure 5·3: The J_{mse} as function of one weight for the linear and non-linear perceptron. The data set contained m=32 samples distributed with statistics given by 5·2. Note that the graph for the linear perceptron is scaled to fit into the same graph.

In this case the weight optimization problem becomes more complex since there is a probability of being trapped in a local minimum or to stop at a random point on a flat surface found in this error space. In feed forward classifiers the number of weights is large due to the hidden layer, the error surface becomes even more complex and it is

very difficult to represent it graphically. Because the non-linear perceptron is contained in the feed forward classifier, it can be concluded that the error surfaces of these classifiers also contain local minima.

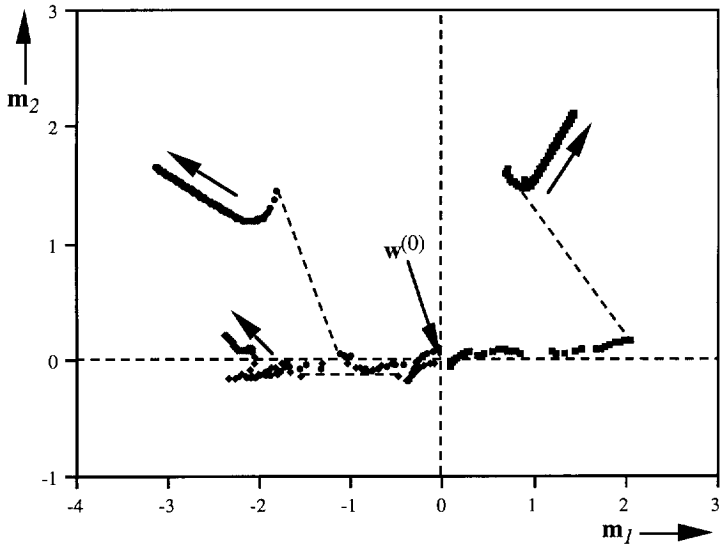

Figure 5·4: The history of three weight vectors found in the hidden units, during 80,000 training sweeps. The arrows show the direction towards the final weights.

Some properties of the weight space can be observed by following the weights during a training session. A network with 3 hidden units (*2-3-1*) is trained and two parameters $\mathbf{M}_{1,i}$ and $\mathbf{M}_{2,i}$ are plotted during training for the three hidden units. In figures 5·4 and 5·5 the histories of changes of these weights are plotted for two different initializations. The learning method is just the standard back propagation method. The initial weights were chosen to be small random numbers.

It is observed that the lengths of all three vectors increase as the number of training sweeps increases. At first the change of the weight vector is irregular but as the training process progresses the behavior becomes more regular. In some segments (denoted by arrows in figures 5·4 and 5·5) a linear change of all three vectors can be observed. Sometimes all weights of one neuron increase proportionally. This is observed between 6000 and 80000 training sweeps for the first initialization (figure 5·4) and between 8000 and 80000 sweeps for the second initialization (figure 5·5).

When the direction of the vectors $\mathbf{M}_i$ practically do not change any more, but only the norm of each vector, then the minimization of the mean square error can be performed in a restricted parameter space. For example, a simultaneous change of all three norms of the weight vector obtained after 16000 training sweeps in the first initialization, reduces J_{mse} from 0.01117 to 0.00855. This was performed by modifying

the back propagation rule to change the norms of the vectors $\mathbf{M}_i$ only and the final mean squared error was obtained in 2000 sweeps.

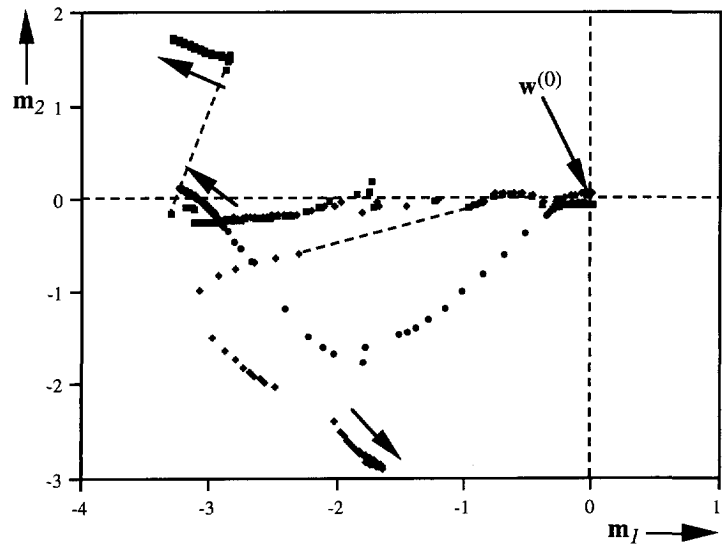

Figure 5·5: The history of three weight vectors found in the hidden units, during 80,000 training sweeps. The arrows show the direction towards the final weights. This figure differs from 5·4 in the chosen initial weight vectors.

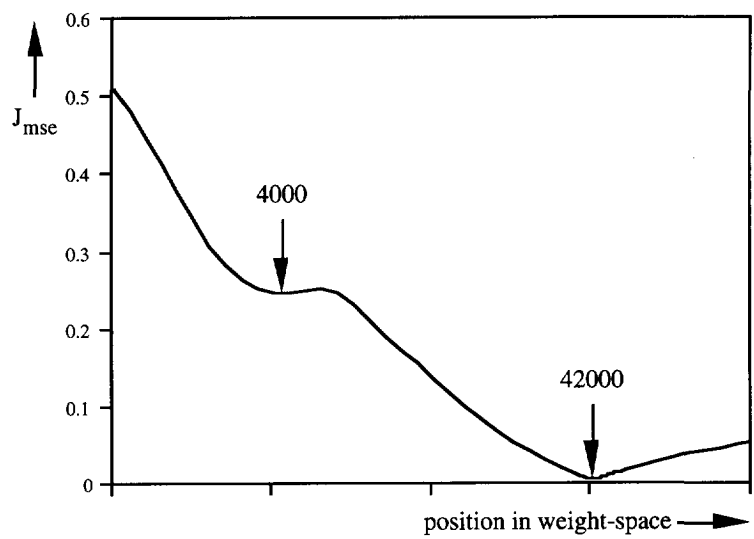

Figure 5·6: A cross section of the J_{mse} criterion, obtained by connecting a line between two pairs of solutions, that are found after 4000 and 42000 training sweeps respectively.

The standard back propagation with all network parameters required an additional 15000 training sweeps to reduce the J_{mse} to 0.00855. It is worth mentioning that for the optimization of the three scale parameters, it was experimentally observed that the back-propagation training algorithm could be used with a 4000 time larger value of η. This is probably caused by the fact that the curvatures (second derivatives or λ_{max}) are much smaller for the scale directions than for all directions of vector $\mathbf{M}_i$. The condition for stable convergence 4·20 predicts that larger values for η are possible if λ_{max} becomes smaller.

The difference between figure 5·4 and 5·5 clearly illustrates the sensitivity of the generalized delta rule (BP) to the initial weights. In figure 5·1 it was shown that the dynamical-behavior differs significantly from one initialization to another, but these two plots show that the final weights are also different.

A cross section of the high dimensional weight space is shown in figure 5·6. This section is obtained by connecting a straight line between two weight solutions that were obtained after 4,000 training sweeps and 42,000 training sweeps, of the same training session. It is observed that there is no straight monotonic descending path from the point A to the point B. There even exists a *hill* on that path. The J_{mse} decreased steadily during the training from point A to B. This fact as well as a non-linear change of the weights shown in figures 5·4 and 5·5 indicates that error *valleys* are neither straight nor convex.

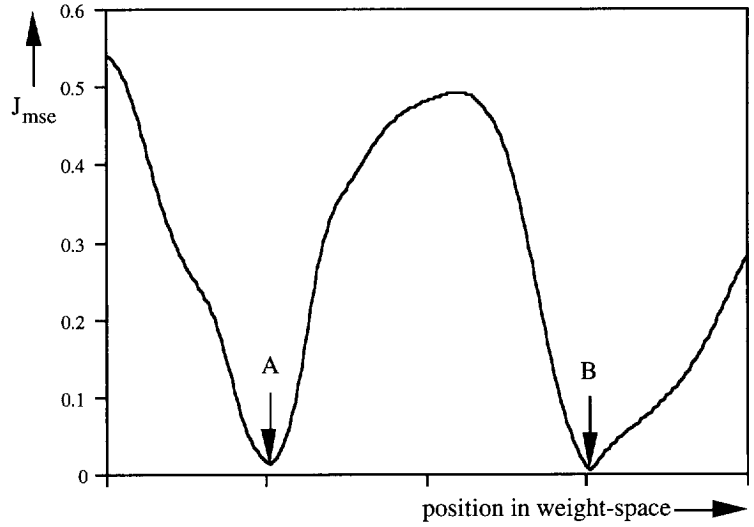

Figure 5·7: A cross section of the J_{mse} criterion, obtained by connecting a line between two pairs of solutions, that are found after 10000 training sweeps. The solutions A and B correspond to two different initial weights.

In figure 5·7 a similar cross section of the multivariate error space is shown. This cross section is determined by two different weights vectors obtained after 10,000

training sweeps which were found by starting from *different* initial points (initializations A and B). There is a huge *mountain* between two *minima* and there is no clear *path* from one *minimum* to the other. To travel from the left minimum to the right one, it is probably easier to restart training from a new initialization, instead of trying to find the path in the complex shaped area.

Due to the flat areas and the complex shape of the error surface the back-propagation training is slow and the training process finishes at different solutions. A different solution is obtained if the training is started from a different initial condition. It is incorrect to assume that the training is always stopped at a local minimum. In the following it will become clear that the back propagation training suffers from local minima and saddle structures in weight space.

5·5 Factors that influence the error surface

The random initialization of the weights of the feed forward classifier often leads the training to different solutions. This is partially caused by the complex shape of the error surface and it is therefore important to know the factors that influence local minima and flat areas in this space. These factors are investigated in this section.

5·5·1 Pattern probability distributions

Suppose that the two classes are tightly clustered around two distant centers. A good classification rule will be a linear classifier because a simple perceptron is complex enough to solve this classification problem. No local minimum is expected to arise from the pattern distributions in this case.

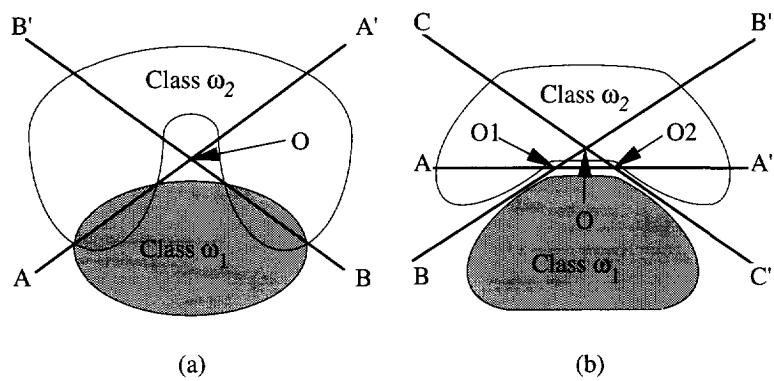

Figure 5·8: Decision boundaries produced by a feed forward classifier with two units in the hidden layer and hard limiting output functions.

If the pattern distribution becomes more complex, for example two classes that are *not* linearly separable such as depicted in figure 5·8a, then a network with two neurons

in the hidden layer will form a piece wise linear decision boundary AOB. This decision boundary is composed from segments made by two hidden layer boundaries AA' and BB'. Because the weight vectors of the hidden units can be exchanged, it can be expected that the error function J_{mse} will have 2 minimum points and that both of them are global.

In the case of the data depicted in figure 5·8b, a trained classifier with two neurons in the hidden layer will produce decision boundaries either BO_1A', AO_2C' or BOC'. These can be formed by the following six combinations of the linear boundaries of two hidden layer neurons:

B'B	A'A
A'A	B'B
A'A	C'C
C'C	A'A
BB'	CC'
CC'	BB'

In this last case it can be expected that the J_{mse} will have six minima of which four are local. The examples above show explicitly that local minima arise easier in the complex structured data case than for linear separable problems. This is related, however, to the structure of the classifier. In this last example, the local minima may vanish if a network with three hidden units is used instead of only two hidden units.

In chapter 3, section 3·5·3, it was shown for the perceptron that local minima can occur. In this example the sub-optimal solution is a result of the pattern distribution and, because the multi-layer perceptron contains the simple perceptron as model, local minima can arise as a result of the pattern distribution.

5·5·2 The complexity of the feed forward classifier

The pattern error function of the simple linear classifier (perceptron for example) has only one minimum. The increase in the number of layers and neurons creates the possibility to interchange the weight vector of the neurons introducing a symmetry in weight space. This creates many equivalent global and local minima. The depth and the number of the local minima depend on the complexity of the classifier. An extreme example is the use of the linear classification function in a case where the number of inputs exceeds the number of training samples. This results in a large number of linear decision boundaries that are all global minima because they all classify the training set without any error.

This last example is a clear example of a classifier with more *complexity* than needed. However, the complexity of weight space also increases even if the complexity of the classifier has not yet reached the required complexity. The number of stationary points monotonically increases with the increase in the complexity of the classifier and thereby the complexity of the pattern error function increases monotonically.

Start for example with a simple feed forward network with one hidden layer, one unit in the hidden layer and one output unit. The mathematical formulation of this network can be written as:

$$g_1(\mathbf{x}) = f(\mathbf{w}_1 \cdot f(\sum_{j=1}^{d} \mathbf{M}_{1,j} \cdot \mathbf{x}_j + \mathbf{M}_{1,d+1}) + \mathbf{w}_2) \tag{5·6}$$

Assume that this network has a minimum for the objective function given by equation 4·7 and for a certain classification problem (data set). Let the weight vector where this point is found be given by:

$$\theta_1 = (\mathbf{M}_{1,1},\ \mathbf{M}_{1,2},\ \mathbf{M}_{1,d+1},\ \mathbf{w}_1,\ \mathbf{w}_2)^T \tag{5·7}$$

Because this is a minimum it is also a stationary point:

$$\frac{\partial \mathbf{J}_{sse}}{\partial \theta_1} = \mathbf{0} \tag{5·8}$$

If the classifier complexity is increased by adding a new unit in the hidden layer, equation 5·6 will become:

$$g_2(\mathbf{x}) = f(\mathbf{w}'_1 \cdot f(\sum_{j=1}^{d} \mathbf{M}'_{1,j} \cdot \mathbf{x}_j + \mathbf{M}'_{1,d+1}) + \mathbf{w}'_2 \cdot f(\sum_{j=1}^{d} \mathbf{M}'_{2,j} \cdot \mathbf{x}_j + \mathbf{M}'_{2,d+1}) + \mathbf{w}'_3) \tag{5·9}$$

The total weight vector for this model now becomes:

$$\theta_2 = (\mathbf{M}'_{1,1},\ \mathbf{M}'_{1,2},\ \mathbf{M}'_{1,d+1},\ \mathbf{M}'_{2,1},\ \mathbf{M}'_{2,2},\ \mathbf{M}'_{2,d+1},\ \mathbf{w}'_1,\ \mathbf{w}'_2,\ \mathbf{w}'_3)^T \tag{5·10}$$

The classifier formulated by equation 5·10 holds all possible classifiers of model 5·6, just by choosing the weights:

$$\begin{aligned} \mathbf{M}'_{1,j} &= \mathbf{M}_{1,j} \\ \mathbf{M}'_{2,j} &= \mathbf{M}_{2,j} \\ \mathbf{w}'_3 &= \mathbf{w}_2 \\ \mathbf{w}_1 &= \mathbf{w}'_1 + \mathbf{w}'_2 \end{aligned} \tag{5·11}$$

Equation 5·10 reduces to 5·6 if the parameters are chosen according to equation 5·11. Because the vector defined in formula 5·7 is a stationary point of the objective function, the immediate result is that this induces a line of stationary points in the weight space of the higher order model. The equation of this line with stationary points is thus given by 5·11.

The stationary points that are induced by a lower model neural classifier will become saddle ridges in a higher model if this is not the global minimum for this model. If the feed forward model is again expanded with a new neuron in the hidden layer (to three), all the stationary points of any lower order model will be found as lines or hyper planes of stationary points in the error surface of this model.

From this analysis is clear that the number of saddle points, global and local minima increase with the order of the neural network model used. This implies that the steepest

descent method will have more difficulty in finding a solution for more complex classifiers. It is an academic question if the training is trapped in a local minimum or suffers from a high dimensional saddle structure; both structures cause the training to slow down or even stop.

5·5·3 The training sample size

When the number of training samples is small there is much empty space between the training vectors in error space. Therefore it is easier to find a good discriminating boundary between the training vectors. It can be expected that in the small training sample case the values of the objective loss function will be smaller and the local minima will be deeper than in the large training sample case.

This intuitive guess is confirmed by numerous empirical studies. The influence of the number of training samples on the shape of the loss function near a minimum (as depicted in figure 5·7) has been analyzed. It was found that often minima become less deep when the number of training samples increases.

[Lee 1991b] performed the back propagation training for *different starting weights* and found that the variance of the classification error is larger for small sized training sets. For example the standard deviation of the training set error decreased from 0.0043 for 50 samples to 0.0025 for 450 training samples. On contrary [Le Cun 1989] performed training of a complex multi-layer feed forward classifier with more than 7000 training samples and reported no problems arising from local minimum effects.

5·6 Training a feed forward classifier

The design considerations listed in section 5·3 are unfortunately not independent of each other. Almost all choices have implications, although some are stronger and some are weaker. The dependencies are also related to the task to be learned and finding a working solution is almost an *art* [Weiss 1991]. It is impossible to optimize all selections independently because the number of combinations is far too large.

The complex error surface, as discussed in section 5·4, shows that the learning can be trapped in many local minima or areas were the mean squared error is only slowly-varying. Different solutions, obtained from different initializations, seem to be connected to each other in a complex way, see figure 5·7 for example. Even the path in weight space can be very complex as depicted in figure 5·6.

The idea is introduced here that learning is a stochastic process and one learning session is just a realization of that process. This idea is closely related to the work reported by [Kolen 1991] dealing with the sensitivity of back propagation to the initial weights. The sensitivity of the training method and the complexity of the error surface make it almost impossible to predict the outcome of the training algorithm. From this point of view the learning process is a random process and should be analyzed as such.

It is important to note that any change to the learning algorithm or network alters the random learning process and a different distribution of J_{mse} can be expected. The question is now: what happens to the probability distribution describing the learning process? What are the consequences for the generalization capabilities ($\hat{\varepsilon}_t$) of the final obtained classifiers if the training algorithm is slightly changed.

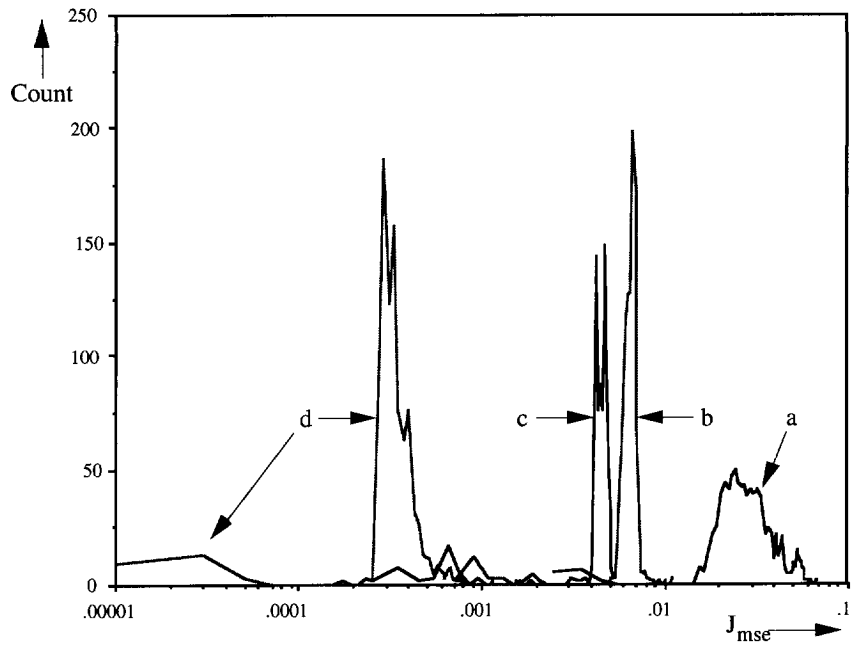

Figure 5·9: Histogram of J_{mse} for different number of training sweeps, based on 1000 initializations. A feed forward classifier 2-3-1 is trained using m=16 samples with distributions given by equation 5·2. The generalized delta rule 4·13 is used with η=0.2, α=0. Graphs a, b, c and d are trained with 1000, 4000, 20000 and 200000 sweeps respectively.

A commonly used heuristic is to train a number of networks, starting from different initial points and to select the *best* network from this set. This heuristic is meant to make the training more robust against the stochastic fluctuations. It seems to favor classifiers with higher generalization error in some situations.

A few design considerations are analyzed in this section and the effect on the generalization capabilities of feed forward classifiers is also discussed. It is impossible to analyze all choices listed in section 5·3, considering the correlation that exists between them. One should not forget, however, that training a feed forward classifier is a high-dimensional optimization problem and that any choice will influence the training.

5·6·1 The probability distribution of J_{mse}

The final J_{mse} is measured for a large number of random initializations. The J_{mse} value for each initialization can be assumed to be a realization of a random variable. It will vary in an interval $[\min(J_{mse}), \max(J_{mse})]$ and the probability density function will generally depend on the way the initial values of the weight vector have been chosen.

The number of training sweeps or cycles used to train the neural network has a significant influence on the distribution of the J_{mse} values. An increase in the number of training sweeps will generally narrow the interval $[\min(J_{mse}), \max(J_{mse})]$ of the distribution. In a case of *perfect* training, which means converging to a unique minimum, most values of the mean squared error will lie close to each other.

In figure 5·9 four distribution density functions of the J_{mse} are presented, obtained from 1000 initializations. The density functions differ in the number of sweeps used to train the classifier. Only one set of training patterns and the same 1000 initial weight vectors were used to obtain all 4 histograms.

It is obvious that the number of training sweeps is a very important factor in the shape and location of the density function $f(J_{mse})$. In most cases the densities are not uni-modal. It indicates that for this particular set of training patterns and architecture there are at least two different local minima. A more detailed analysis of the weight vectors obtained after 200,000 training sweeps has shown that the weight vectors and decision boundaries corresponding to one mode in the histogram are different (see two arrows in figure 5·9 for graph d) from the other mode. For the mode with a peak at 0.00023 the weights are close and can be supposed to belong to the same minimum.

To obtain a small value of the objective function J_{mse} the use of one initialization and many training sweeps is not sufficient. The combination of several initializations, together with a sufficiently large number of training sweeps will increase the probability that a solution with a low value of J_{mse} will be obtained.

Two open unsolved questions remain: when to stop to the training procedure and how large should the number of initializations be chosen? It is difficult to provide a definitive answer to these questions before the neural network training process begins. It is probably true that the increase in the number of training sweeps can not replace the number of initializations and vice versa.

5·6·2 Selecting the *best* classifier from more initializations

Let us train n neural network classifiers each time starting from a different initial weight vector and use the same set of training patterns for all initializations. The final mean squared errors J_{mse}, for all n classifiers will be measured. Suppose now that a very large set of test patterns is available to evaluate all these n classifiers. Each feed forward classifier can be characterized by the following vector:

$$(J_{mse},\ \hat{\varepsilon}_t)^T \tag{5·12}$$

In figure 5·10 an example is given of the distribution of 1000 pairs of these vectors, corresponding to 1000 trained feed forward classifiers. Each classifier is trained with the same data and only differs in the random starting point. The standard back-propagation training algorithm 4·13 was used with η=0.3, α=0.1 and 5000 training epochs. The data set distribution and network architecture is described in section 5·2. The number of training samples is m=160 (80 per class). The number of samples to estimate the generalization error $\hat{\varepsilon}_t$ was 2000 (1000 per class).

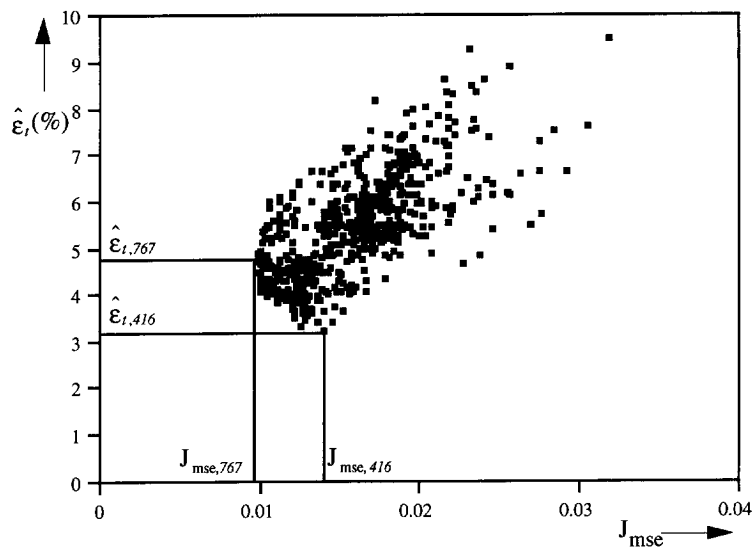

Figure 5·10: A scatter diagram of the mean squared error versus the generalization error. The data points correspond to a trained network using the same learning data but starting from different initial weights.

In selecting the *best* network, the normal thing to do is to use the estimates of the mean square error, which is the equivalent of the re-substitution error normally used in statistical pattern recognition. The best initialization in mean squared sense was trial number 767 which resulted in $J_{mse,767}$=0.0098. The true error, the generalization error $\hat{\varepsilon}_t$ of this classifier, appeared to be $\hat{\varepsilon}_{t,767} = 4.7\%$. From the scatter diagram we see that the best initialization with respect to the generalization error $\hat{\varepsilon}_t$, was number 416 with $\hat{\varepsilon}_{t,416} = 3.2\%$. Thus an inexact selection of the *best* initialization, increased the generalization error by 47%!

In section 2·4·7 the selection of a pattern classifier was discussed and it was explained that an increase in classification error can occur due to the selection of a sub-optimal classifier. The increase in generalization error due to a non-ideal selection of the *best* network from a number of initializations is the same type of phenomenon but now appears to be critical in neural networks.

The increase in the generalization error depends on the set of vectors defined by equation 5·12. This set depends on the architecture of the neural network classifier, the

distribution density function of the pattern vectors, the actual set of training pattern vectors used to obtain the estimates J_{mse} and, of course, on the training procedure used.

5·6·3 Correlation between J_{mse} and ε

If a classifier is only trained for a few training epochs then the J_{mse} will be comparatively large and will vary in a wide interval. It was shown in the previous section (figure 5·9) that the interval becomes narrower and shifts to the left as the number of training sweeps increases. There is no reason to expect that when J_{mse} is large, a classifier with a small generalization error will be obtained. To the contrary, when J_{mse} is large then the generalization error is large too and when J_{mse} is small, one can *hope* to obtain a small value of $\hat{\varepsilon}_t$.

When a feed forward network is trained insufficiently, one can expect a wider distribution of J_{mse}-$\hat{\varepsilon}_t$ and a large positive correlation between them. Assume now that in each training session the classifier is trained sufficiently using a large number of training sweeps. In this situation a classifier from the very left segment of figure 5·10 is likely to be obtained and a lower correlation between J_{mse} and $\hat{\varepsilon}_t$ is to be expected.

This deduction is confirmed by experimental observations. In figure 5·11 four distributions J_{mse}-$\hat{\varepsilon}_t$ are shown, that correspond to the same experimental conditions as the histograms presented in figure 5·9. With an increase in the number of the training sweeps, two clusters become clear. A more detailed analysis of the weight vectors obtained after 200000 training sweeps indicate that the weights that correspond to the left cluster produce different decision boundaries. These solutions correspond either to different local minima or to different parts of a flat area (or saddle structure) of the pattern error function. All the weight vectors that correspond to cluster on the right side produce a similar decision boundary so it can be concluded that they correspond to one *local* minimum.

The correlation between J_{mse} and $\hat{\varepsilon}_t$ is important when selecting one classifier from a number of different initializations. A strong *positive* correlation ρ will ensure that the classifier with the lowest mean squared error is likely to be the classifier with the lowest probability of error. When a strong *negative* correlation exists between J_{mse} and $\hat{\varepsilon}_t$, then selecting the classifier with the lowest mean squared error will favor the classifier with a *larger* generalization error. A correlation that is approximately zero suggests that repeating the training with different initial conditions, will not improve the classification performance of the final classifier.

In figure 5·10 the selection of the best classifier from 1000 initializations resulted in an increase in generalization error. The measured correlation was ρ=0.69 and even in this case, with a relatively strong correlation, the number of initializations was too large. There exists an optimal number of initializations that on *average* will favor the classifier with the lowest generalization error. The optimal number can be calculated from the distribution shown in figure 5·10 and an efficient algorithm to do this is explained in [Raudys 1992].

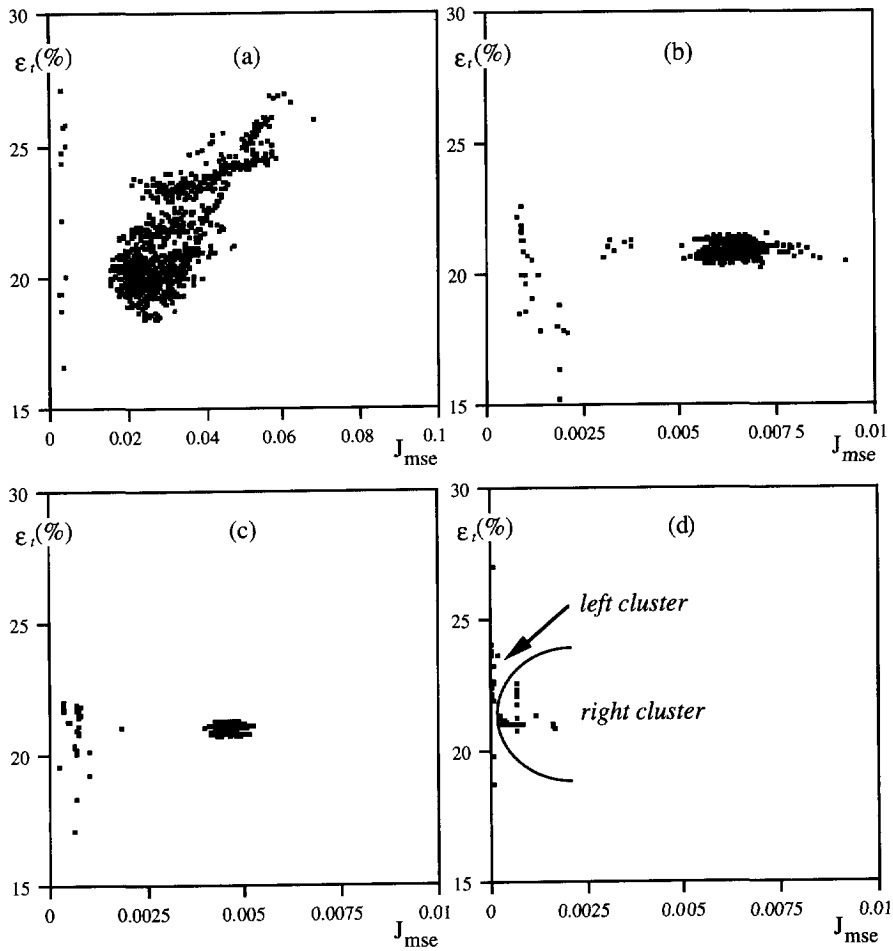

Figure 5·11: Four scatter diagrams of J_{mse} versus generalization error obtained for different numbers of training sweeps. The experimental settings correspond to figure 5·9. The number of training sweeps used are respectively: (a) 1000, (b) 4000, (c) 20000 and (d) 200000.

The theoretical understanding of this phenomenon has been presented in section 2·4·7 which shows that the training sample size is in this case not enough to select one classifier from a large set of classifiers. One can even obtain a strong negative correlation between J_{mse} and $\hat{\varepsilon}_t$. Such a situation is observed in figure 5·12 where a *2-3-1* network was trained using 20000 training sweeps and $m=80$ samples. The correlation for this example was ρ=-0.78. In this situation the training data permits to apply only one initialization.

In section 5·3·10 it was explained that using a limited number of training cycles is related to the regularization method described by equation 5·3. This suggests that the

correlation ρ dependends on the number of training sweeps used. For the experiments found in figure 5·11 the following correlation's correspond to the individual graphs: ρ_a=0.71, ρ_b=0.31, ρ_c=0.06 and ρ_d=-0.22.

This indeed confirms the theoretical results from [Sjöberg 1991] and clearly shows that the optimal number of initializations dependends on the number of training cycles. These dependencies that exist between the different design considerations make it difficult to select an optimal setting for all design considerations.

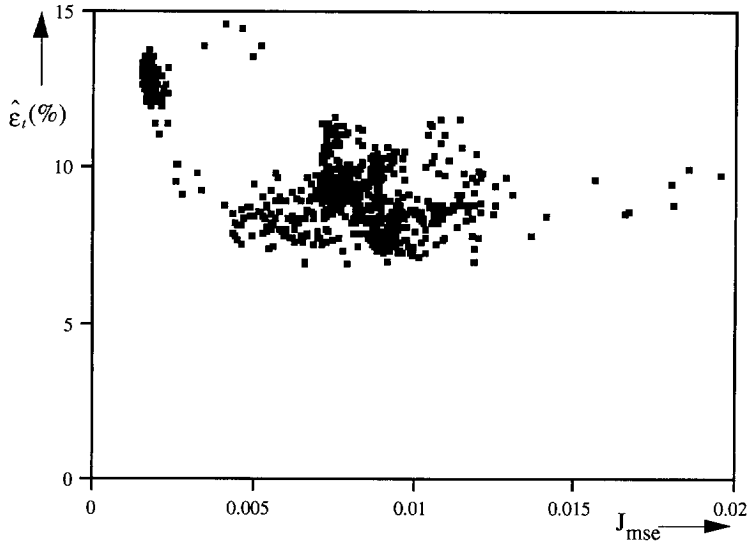

Figure 5·12: A scatter diagram of J_{mse} versus ε obtained for a training set of m=80 samples and 20000 training sweeps. A strong negative correlation ρ=-0.78 is found for this example.

The value of the correlation depends on how well the training pattern vectors reflect the general population. When training patterns have a different structure than the general population, one can expect the correlation will be small or even be negative.

The set of training patterns is usually a random selection from the universe. It can therefore be concluded that the correlation is a random variable as well. In figure 5·13 different scatter diagrams are shown for different sets of training data. Again a feed forward classifier with a *2-3-1* architecture is trained with a training set containing 80 samples and using 4000 training sweeps.

The distributions in figure 5·13 are indeed different. This supports the observation that individual peculiarities of the particular set of training samples play an important role. The four correlations that are found in this figure are, ρ_a=-0.10, ρ_b=0.88, ρ_c=0.85 and ρ_d=-0.21. It should be noted, however, that the two positive correlations are dominated by a few outliers.

When a negative correlation exists between J_{mse} and $\hat{\varepsilon}_t$, a reduction of the optimized criterion will cause an increase in the generalization error. This phenomenon occurs

when the classifier is too adapted to the training data. This is called *over-training* in neural networks literature. The value of the correlation between these criterions could probably serve as a measure for this qualitative phenomenon.

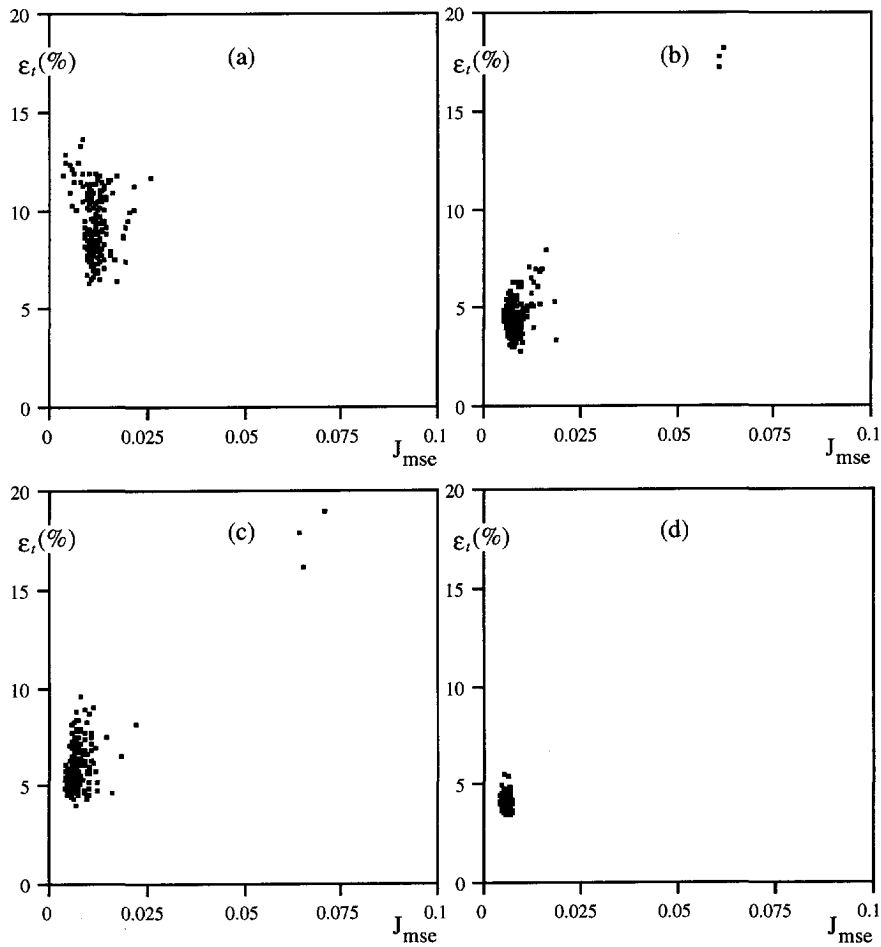

Figure 5·13: Four scatter diagrams of Jmse versus e, obtained from four different training sets (m=80). All networks are trained using back propagation with 4000 training sweeps. The correlations for these experiments are respectively: (a) ρ=-0.10, (b) ρ=0.88, (c) ρ=0.85 and (d) ρ=-0.21.

The distribution of J_{mse}-$\hat{\varepsilon}_t$ depends on how well the training samples represent the statistics of the general population. This is impossible to predict without some additional information. It is impossible to predict from the training samples only, if the correlation will be positive or negative. This is a well-known fact in classical statistics: it is impossible to predict from a learning set if the true mean squared error will be larger or smaller than the mean squared error measured on the learning set.

The only conclusion that can be derived from these results is that one should use a sufficiently large number of training samples, such that the increase in the generalization error caused by the non-ideal selection of the best initialization will be sufficiently small. In [Raudys 1981, 1987] the influence of the sample size on the model selection in pattern recognition has been investigated. He shows that the increase in the classification error due to improper model selection is of the order of one standard deviation of the re-substitution estimate $\hat{\varepsilon}_r$ used to choose the best initialization. The variance of this estimate is given by equation 2·16 and when $\hat{\varepsilon}_r << 1$ then the recommendation follows:

$$m >> \frac{1}{\hat{\varepsilon}_r} \tag{5·13}$$

The above equation can be used to determine the number of training samples necessary to avoid (possibly only to detect) unpleasant effects due to over-training. The number of training vectors determined from the above inequality is *not* the number of the samples necessary to train a neural classifier with good generalization properties. This depends on the complexity of the classifier, the number of inputs d and the number of neurons in the hidden layer h, as well as on a complexity of the pattern space see, [Raudys 1991].

5·6·4 The influence of different design considerations

In the previous sections the influence of the peculiarities of a specific data set, the number of training epochs and the random initial state was shown. The design considerations of section 5·3 all influence the training. An investigation of all these practical issues on the expected probability of error is too complex because of the correlation that exists between them.

It has become clear that any change to the algorithm and the training paradigm should be introduced with care. To design a high performance classifier, a classifier that is close to Bayes optimal, the influence of different choices on the expected probability of error must be investigated. The only way to do that is to use a different test set to find the correlation between J_{mse}-$\hat{\varepsilon}_t$. This is surely a practical limitation of the applicability of neural networks as classifiers.

5·7 Discussion

This analysis confirms the deductions of previous authors, that the surface of the pattern error function in the multivariate weight space is very complex. It has many local minima and nearly flat areas or saddle structures where the gradient of the error function is extremely small. The training algorithm stops somewhere at a random point due to the complex weight space and random chosen starting point.

The conclusion from other authors, [Kolen 1991] for example, that back propagation is very sensitive to the initial weights is confirmed. The result of the training, the weight vector, is a random variable and the statistics of this distribution are heavily dependent on the design considerations listed in section 5·3.

The weight initialization plays an important role in the training of a feed forward classifier and one initialization is often not sufficient. To be robust against outliers, one must perform several training sessions, each time beginning from a different initial weight vector. The number of initializations needed to find, on average, a good classifier, is heavily dependent on the learning rate η and number of training sweeps. An excessive increase in the number of initializations or training sweeps does not always lead to a better classifier.

There is no way to predict the usefulness of an increase in the number of sweeps or an increase of the number of initializations. The only recommendation is to use a sufficiently large number of training patterns. A proposed guess for this amount of samples is given by equation 5·13. This last result will only ensure that the increase in the generalization error caused by the non-ideal selection of the best initialization will be sufficiently small. It does not give any guaranty on the generalization capabilities of the neural classifier.

In *classical* statistical pattern recognition it was already observed that a classifier trained with a finite data set shows a dependence on the peculiarities of a specific data set. When the training sample size is small, these fluctuations are large and the classifier performance will decrease as a result of the finite sample size (figure 2·3). The feed forward classifier suffers from this problem but also shows a dependence on a second stochastic component. It is new to pattern recognition: random perturbations of the training method have a significant influence in the design of a classifier. This is a very undesirable feature of feed forward classifiers.

The correlation between the mean squared error (J_{mse}) and the generalization error is an important quantitative measure. If a strong negative correlation exists between these two, the initialization with the lowest J_{mse} will lead to the selection of the worst classifier. This suggests that the network complexity and training do not match the data complexity and the number of training samples. This can arise due to a peculiarity of a training set or be more systematic for a certain distribution and network choice. The correlation is suggested as a quantitative measure of this phenomenon that is often called *over-training* in neural networks.

A possible explanation for a correlation between the optimized criterion and its true value can be found in figure 5·14. In this figure a typical learning curve is shown as function of the number of training cycles. The optimized criterion, in network training the mean squared error, normally decreases steadily as function of the number of training cycles. The test-set or generalization error is expected to decrease but at some point during training, point B in figure 5·14, it can increase. This point is where over-

training is starting to occur and the feed forward classifier is too adapted to the finite training data.

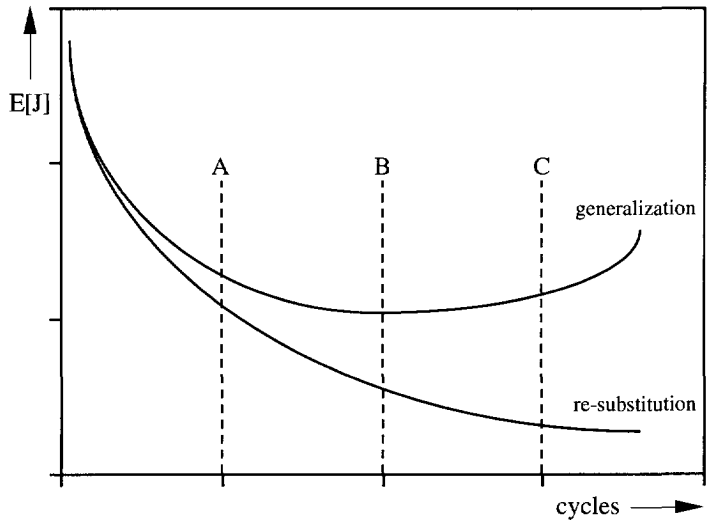

Figure 5·14: A typical graph showing the optimized (re-substitution) and true (generalization) value of the optimized criterion as function of the number of training cycles.

Assume that a network is trained with a number of training cycles that correspond to point A in figure 5·14. The initial weights will influence the final mean squared error where the training will stop. This causes a random fluctuation with mean value corresponding to point A and some variance. At this particular point the derivative of the generalization error with respect to the re-substitution estimate is positive. If the variance caused by the initial weights is relatively small then it can be concluded that a positive correlation will occur for an ensemble of networks. In figure 5·10 an example of such an ensemble is (experimentally) observed.

A similar conclusion can be derived for point B but the derivative of the generalization error with respect to the re-substitution estimate is zero here. Now a zero correlation is expected to be measured for a collection of networks. Finally, if the number of training sweeps is increased to point C, then the average generalization error will start to increase. A collection of networks will now clearly have a negative correlation. The observed correlations of the four scatter diagrams of figure 5·11 agree with this hypothesis.

An observed strong positive correlation suggests that the collection of classifiers was trained with an insufficient number of training cycles. At this point it seems to be better practice to increase this number and not to increase the number of initializations.

6
Generalization and Network Complexity

6·1 Introduction

When analyzing the multi-layer feed forward networks from a statistical or parameter estimation point of view, it is obvious that these systems have a large number of parameters. Systems with large numbers of free parameters, however, do not perform very well in parameter estimation problems because the convergence to a proper set of values of these parameters requires a very large number of training examples. In the parameter estimation literature, it has been shown that the variance in the parameters increases when the model contains more parameters (the Cramér-Rao bound, see [Mood 1974]), which results in poor performance (see [Van den Bos 1982]).

A related phenomenon is the so called *peaking effect,* where the overall performance of a pattern recognition system decreases when the number of features or the number of parameters increases (figure 2·3). More relevant features should imply a better classification performance but because the parameters of the classifier are determined by a finite and probably too small data set, the performance measured on a test set decreases at some point (see [Duin 1978] or [Vapnik 1982]).

The same conclusion derived in the field of parameter estimation and statistical pattern recognition, that the number of free parameters should be reduced as much as possible, lead researchers in both fields to be very skeptical about developments in neural networks. In the neural network literature only a few researchers have been concerned about the large number of parameters that are trained. The *weight sharing* technique by [LeCun 1989] is a good example of reducing the number of parameters to improve the network generalization capabilities.

Despite the bounds formulated by the statistical laws, a number of reported neural network applications claim to be successful. A good example is the NETtalk experiment of [Sejnowski 1986]. Here 25946 weights are estimated with only ≈5000 training examples with a reported performance of the network classifier of $\hat{\varepsilon}_t$=20%. It seems unrealistic from a classical point of view that generalization can be expected from a system where the number of free parameters exceeds the number of training samples. This particular example will be investigated in detail in the next chapter because the data set is available.

The NETtalk example is not a unique case; similar results can be found in many conference proceedings where applications of feed forward classifiers are published. In

table 6·1 a list of examples is enumerated to show that there is a fundamental difference between neural network literature and traditional guidlines concerning the minimum required sample size.

Table 6·1: Some applications of feed forward classifiers from literature, with remarkable sample sizes from a traditional point of view.

Application	Weights N	Samples m	Performance $\hat{\varepsilon}_t$	Reference
Text to speech translation	>25,000	≈5,000	≈20%	[Sejnowski 1986]
Sonar target recognition	1105	192	15.3%	[Gorman 1988]
Control of autonomous car	>36,000	1,200	Car drives on winding road	[Pomerleau 1989]
Back-gammon player	>11,000	3,000	Computer champion	[Tesauro 1990]
Sex recognition from faces	>36,000	90	9.1%	[Golomb 1991]
Character recognition	9,900	5,000	5.5%	[Sato 1991]
Landsat remote sensing	1,800	50	5-10%	[Kamata 1992]
Signature verification	480	280	5.4%	[Sabourin 1992]

In this chapter the small sample size properties of feed forward classifiers will be investigated. In section 6·2 a new theory on sample size and generalization will be discussed. This theory is unfortunately not useful for explaining any of the results listed in table 6·1. An intuitive explanation is given in the next section, where it is shown that a large redundancy exists in the weight space. In section 6·4 the classification capacity of feed forward classifiers is experimentally investigated by performing an analysis similar to the approach of [Foley 1972]. The conclusions from this section are experimentally verified in section 6·5 where small sample size behavior is observed for a number of distributions.

6·2 A bound on poor generalization

In this section a theory about sample size and expected probability of error will be discussed. This theory was published in the Russian literature in the early 1970's but became known to *western* researchers much later (1982). It was mainly developed by two Russian researchers and is known as the *Vapnik-Chervonenkis theory*. Here only the basics will be explained and a detailed treatment can be found in [Vapnik 1982] or

[Natarajan 1991]. In [Kraaijveld 1993] the Vapnik-Chervonenkis theory is applied to feed forward classifiers and the consequences and limitations are discussed there in detail.

6·2·1 The Vapnik-Chervonenkis theory

A classifier g(), that is trained for a two-class problem can be considered as a Boolean function. For a family or set of classifiers G, the training procedure is the selection of one Boolean function from a larger set. The selection of the *best* classifier from a larger set of classifiers was already discussed in section 2·4·7 and the following key result is an extension of formula 2·26 ([Vapnik 1982]):

$$P(\underset{g\in G}{MAX}\, |\hat{\varepsilon}(g) - \varepsilon(g)| > \Delta\varepsilon) \leq 6 S_G(2m) e^{-(\Delta\varepsilon)^2 m/4} \tag{6·1}$$

The left-hand side of this equation is the probability that the measured error $\hat{\varepsilon}(g)$ of any classifier from the set of all considered classifiers G, will deviate more than $\Delta\varepsilon$ from its true probability of error $\varepsilon(g)$. The right-hand side is a bound on that probability and involves a *growth* function $S_G(m)$. The function $S_G(m)$ is the maximum number of different binary function that can be implemented by G on any set of m samples.

The growth function is specific for a family of classifiers and it is not always obvious how to determine this function. For the linear classifier this function is given by [Cover 1965] and is simply obtained by multiplying fraction 3·58 by 2^m, the total number of binary functions for m points:

$$S_{lin}(m,d) = \begin{cases} 2^m & m \leq d+1 \\ 2\sum_{i=0}^{d} \binom{m-1}{i} & m > d+1 \end{cases} \tag{6·2}$$

For a fixed dimension d, this growth function is exponential until $d+1$ and after this point the function is polynomially bounded. In figure 3·14 Cover noticed that all curves pass through $m=2(d+1)$ and defined this as the capacity of the linear classifier. A second characteristic point of these curves is $m=d+1$, where a transition from exponential in m to a less rapid increase occurs. This observation leads to the following definition:

Definition of capacity [Vapnik 1982]. The capacity V of a set of classifiers G is defined as the maximum number of points that can be given an arbitrary Boolean label by the functions in G.

The capacity V is a single number that characterizes the complexity of a set of classifiers. The set of linear classifier has a capacity of $V=d+1$ according to this definition. The capacity can be infinite but, for a bounded capacity, the growth function is bounded by the following theorem.

Growth function theorem [Vapnik 1982]. The growth function $S_G(m)$ of a set of classifiers G with finite capacity V is at most:

$$S_G(m) = \begin{cases} 2^m & m \le V \\ \le \sum_{i=0}^{d} \binom{m}{i} & m > V \end{cases} \tag{6·3}$$

This bound can be reformulated for $m>V$ by a polynomial approximation [Natarajan 1991]:

$$S_G(m) \le \left(\frac{em}{V}\right)^V \tag{6·4}$$

In this formula e is the base of the natural logarithm. If $S_G(m)$ grows less rapidly than exponential ($m>V$), then the right-hand side of 6·1 can be made arbitrarily small by choosing m large enough. This bound ensures that with a certain probability no element from G deviates more than $\Delta\varepsilon$ from its performanceas measured with the m samples. The classifier with the lowest probability of error is normally chosen and because this bound holds for all elements in G, the re-substitution error $\hat{\varepsilon}_r$ and the corresponding true error ε are bounded by this theorem.

The Vapnik-Chervonenkis theorem given by equation 6·1, is a very general approach to the sample size problem. The set of considered functions G may contain any collection of functions as long as the capacity of this set is bounded. This bound is independent of the distribution of the patterns so it can be applied to any (real world) problem.

This theory is closely related to [Cover 1965] for the linear classifier. Cover argued that a classifier should be over-determined such that the *a priori* probability of a learning set being separable is low. The Vapnik-Chervonenkis theory establishes the link between this *a priori* probability and the error estimates $\hat{\varepsilon}_r$ and ε. The sample size can now be chosen such that the desired accuracy is guaranteed.

6·2·2 Applying the Vapnik-Chervonenkis theory

When equations 6·1 and 6·4 are combined, one can calculate an upper-bound for the number of samples needed to obtain a desired generalization. To make the right-hand side of equation 6·1 small, the exponential part $e^{-(\Delta\varepsilon)^2 m/4}$ must be significantly smaller than the growth function $S_G(m)$. The number of training samples which achieves this is largely dominated by the factor $(\Delta\varepsilon)^2$. For small values of $\Delta\varepsilon$ the predicted upper bound dampens any enthusiasm, to be used for practical classifier design [Hertz 1991, page 156]. To illustrate this consider the following case for a linear classifier:

$$\left.\begin{array}{l} d = 20 \\ V = d + 1 = 21 \\ \Delta\varepsilon = 0.01 \\ P(\underset{g \in G}{MAX} \mid \hat{\varepsilon}(g) - \varepsilon(g) \mid > \Delta\varepsilon) \leq 0.05 \end{array}\right\} \Rightarrow \; m > 13 \cdot 10^6 \qquad (6\text{·}5)$$

A 20-dimensional linear classifier trained with $13 \cdot 10^6$ samples, is with probability 0.95 within 1% of it's measured (re-substitution) error. If equation 3·59 or table 3·2 are used to determine a sample size, a bound of m>1000 seems to be more than sufficient for Gaussian distributed patterns, to be within that range of the Bayes classifier.

The difference of four orders of magnitude between the observed sample size and the theoretical bound is caused mainly by the fact that the bound is distribution free. When no assumptions are made about pattern distributions than error counting is the only way to measure a classifiers performance. In section 2·4·1 it was noted that this estimate is a binomial random variable and in figure 2·4 the confidence intervals as function of sample size are shown for the worse case. This figure clearly shows that for small deviations $\Delta\varepsilon$, the required sample size increases dramatically.

The Vapnik-Chervonenkis bound considers all possible classifiers that are induced by m samples (the growth function) and thus requires that all these estimates are within the specified error $\Delta\varepsilon$. In pattern recognition the classifier with the lowest error ($\hat{\varepsilon}_r$) is selected so only this classifier is considered. It is possible that the largest deviation is obtained for another unimportant classifier. This fact is, however, ignored in this bound.

In the previous section it was already mentioned that it is not simple to obtain a growth function for a specific family of classifiers. It is problematic to obtain the capacity V of a feed forward classifier and a bound exists only for networks with TLU's. The following bound is published in [Baum 1989]:

$$V \leq 2N \log_2(eh') \qquad (6\text{·}6)$$

In this formula e is the base of the natural logarithm and h' is the total number of TLU's in the classifier. This bound is unfortunately proportional to the number of weights N and a network with sigmoids is expected to have even a higher capacity. A feed forward network often contains a large number of parameters (table 6·1) so this bound suggests that this type of classifier has a very high capacity.

6·2·3 Discussion of the Vapnik-Chervonenkis bound

The Vapnik-Chervonenkis theory, together with the bound on the capacity of feed forward classifiers, can not explain any of the results listed in table 6·1. The sample size should at least be much larger than the classifier's capacity so that $S_G(m)$ grows less rapidly than exponential and bound 6·1 becomes useful. All examples in table 6·1 violate this minimum requirement.

In [Kraaijveld 1993, chapter 4] a number of possibilities are investigated to explain generalization with a sample size lower than the theoretical capacity of a classifier. The

approach follows the Vapnik-Chervonenkis theory and shows some experimental observations of the capacity of a classifier trained in practice.

In the following sections a different approach is used to obtain experimental evidence that the results listed in table 6·1 are plausible. A redundancy in weight space is shown in the next section for some non-trivial classification problems. In section 6·4 the definition of capacity of a classifier is extended and investigated for some different practical settings.

6·3 Redundancy in weight space

In this section some experimental evidence will be described that indicates why the classification results listed in table 6·1 might be possible. It is shown for some non-trivial classification problems, that a large fraction of the weights of the neural network is of less importance and does not need to be measured with high accuracy. The remaining small fraction of the total number of weights is indeed critical for the classification. The sample size compared to this small fraction of the weight space is more realistic from a classical point of view. This work has been published in [Schmidt 1992].

6·3·1 Training networks with random weights

The approach that will be followed is that the weights of the hidden layer(s) $\mathbf{M}_i$ are chosen randomly whereas the output layer is trained by the matrix-inverse technique of equation 3·31. The experiments show that a near Bayes performance can be reached for a number of non-trivial learning problems. From this it can be concluded that the parameters found in the hidden layer, if they can be chosen at random, do not add very much functionality to the classifier. This redundancy of the solutions in parameter space might indicate the nature of some of the successes in the literature.

I would like to emphasize that the method presented in this section, to train feed forward networks by setting some parameters to random values, is not presented as an *alternative* training method. This method is introduced only to analyse the functional behavior of the networks with respect to the learning.

The training procedure can be summerized with the following steps:

***Step 1**: Set the weights in the hidden units ($\mathbf{M}_i$) to uniform [-1..1] random values.*
***Step 2**: Determine the weights of all the output units with equation 3·31.*

If the values of the input features in the application domain is between [*-1000* and *1000*] then this method can be optimized by setting $\mathbf{M}_i$ to a random value in a more appropriate range. This is, however, not always needed and experiments show that for numerical stability this is sometimes preferred. It does not improve the method dramatically.

Another possibility is to set the weights $\mathbf{M}_i$ to a random subset of the training data set. If not enough data samples are present the remaining units are removed from the network. The weights connected to the threshold are set to one. This last method can be interpreted as setting the hidden units to act as *correlators* for a number of samples in the learning data set.

6·3·2 Experiments and results

The two *random* methods described in the previous section have been compared with the standard back propagation method (BP) for low dimensional problems. The simulations were performed with the artificial neural network library ANNLIB (see chapter 8) and a SUN-4 workstation. For these experiments three different data sets were used that were used earlier. The three data sets are all based on the Gaussian probability distribution and the statistics are given by equations 4·28, 4·29 and 3·55.

For every data set a learning set of 100 samples per class was generated and a test set with 1000 samples per class was generated. In all the experiments *8* hidden units are used (*2-8-1*). This architecture contains 3x8=24 parameters in the hidden layer and 9 parameters in the output layer for a total of 33 trainable parameters.

The weights in the hidden units are either set to random values uniformly distributed between [*-1..1*] (called *unif-random* in table 6·2) or they are set to the input values of a randomly selected subset of the learning set (called *data subset* in table 6·2). The average performance (measured on a test set) is calculated for 100 trials.

Furthermore these data sets are used to train a network with the standard back propagation rule (BP) where the initial weights are chosen randomly from a uniform distribution between [-1..1]. The learning coefficient η and momentum term α are set to 1.0 and 0.9 respectively and the network is trained for 500 epochs. The average performance (determined with an independent test set) is calculated for 100 trials.

Table 6·2: The random hidden weight method compared for different data sets. The average classification error and the standard deviation are measured for 100 experiments.

Data set	Method	$\hat{\varepsilon}_t$ (%)	Std. dev. (%)
(4·28)	Unif-random	21.4	0.27
	Data subset	21.6	0.25
	Back prop.	24.1	4.41
(4·29)	Unif-random	23.0	1.81
	Data subset	23.7	1.79
	Back prop.	20.3	4.91
(3·55)	Unif-random	13.4	1.61
	Data subset	14.7	1.64
	Back prop.	20.3	7.20

The results of these experiments are summerized in table 6·2, where the measured average classification errors of the three methods are shown. The standard deviation of the solutions obtained are shown also and this number is apparently much larger for the back propagation method.

6·3·3 Conclusions from random weights experiments

Table 6·2 clearly shows that the random hidden layer procedure generates better networks for data set 4·29 and 3·55. Data set 4·29 is slightly worse, compared to the standard back propagation method. However, the standard deviations in the experiments are much smaller for both the random methods than for the back propagation method.

From these experiments it can be concluded that the output layer is significantly more important than the weights found in the hidden layer. The 24 parameters in the hidden layer (3x8) are of less importance than the 9 weights found in the output unit. In the 33 dimensional parameter space of the complete network a sub space of 24 dimensions is full of combinations that may be regarded as statistically acceptable solutions of the classification problem.

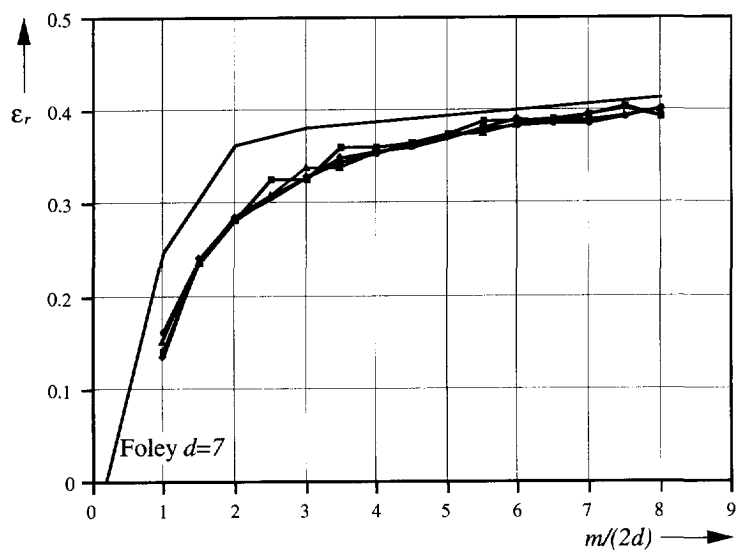

Figure 6·1: Experimental results of the measured average re-substitution error for Fisher's linear discriminant and identical uniform distributions. Graphs for d=3, d=7, d=14 and d=21 are shown. The curve found by [Foley 1972, figure 3] for d=7 is sketched for comparison.

If it is true that this experimental experience holds for larger networks, for example the NETtalk experiment mentioned earlier, then it can partially explain the successes of these experiments. If again the weights found in the output units are of significantly

more importance then the other weights, only ≈2500 (of the total of >25,000) weights need to be measured accurately. The approximately 5000 learning examples are used for this task and although this is still not a very over determined situation, it is certainly more reasonable.

The examples listed in table 6·1 are all classifiers using the feed forward architecture and thus have in common that only a small fraction of the weights is found in the final hidden to output layer. Although the experimental observation described in this section hints that not all weights are important, one should be careful in generalizing this idea to high dimensional problems.

6·4 The capacity of feed forward classifiers

The sample size consideration of [Cover 1965] discussed in 3·6·1 is enhanced with the theory by Vapnik-Chervonenkis (section 6·2). In section 3·6·1 the simulation results by [Foley 1972] were mentioned. These can be regarded as an extension to the initial idea from Cover. In this section these experiments are repeated and applied to feed forward classifiers.

6·4·1 The linear classifier and Foley experiments

The following interesting simulations are performed by Foley. A two-class problem with equal *a priori* probability is considered. Each class has an identical d dimensional class-conditional probability and each of the features is statistically independent and uniform over the interval $[0..1]$. A data set with $m/2$ samples is generated for class ω_1 and ω_2 respectively. Fisher's linear discriminant (see section 4·4·3) is computed and the re-substitution error is measured for a number of different data sets. The sample size is varied for different fractions of sample to feature size.

The results of the average value computed from 100 different data sets are shown in figure 6·1 for dimensions $d=3$, $d=7$, $d=14$ and $d=21$. The confidence intervals are omitted here, otherwise this figure becomes too messy. These graphs do not differ significantly from a statistical point of view. The graphs in figure 6·1 seem to *contradict* with the original results found in [Foley 1972, figure 3] (see the sketched Foley curve) for low dimensions ($d<14$). No confidence intervals are printed in this publication and no information is given about the number of different data sets that are used to measure the figure.

From this figure and the results published by Foley, it appears that the error rate on the design set is very misleading for cases where the ratio $m/(2d)$ is small. The true error rate of this data set is in this case 0.5. The difference between the measured value ε_r and 0.5 is thus an estimation for the bias due to the finite sample size. The curves level off for ratios larger than ten [Foley 1972] and the rule-of-thumb of section 2·4·6 is based on this observation.

The experiment of [Foley 1972] is another way to quantify or judge the relation between sample size and classifier complexity. With this approach some notion is obtained about the scaling relationship that exists between these two. The key idea is to use a distribution with no discriminative power and measure the re-substitution error as a function of sample size.

A complex classifier is a classifier that can classify a larger fraction of an arbitrarily labeled data set correctly. This new definition differs from the definition of complexity by Vapnik that involves only one number. The advantage is that this definition of complexity is based on an average capability and is not a bound to the possible maximum capacity. A graph such as figure 6·1 is a *fingerprint* of a classifier and this idea can be applied to any type of classifier with finite capacity V.

A weak point in this definition is the use of some specific distribution in the evaluation, as in the two identical uniform distributions used by Foley. In figure 6·2 the experiment of Foley is repeated but now using two identical Gaussian distributions with zero mean and unit variance. In this graph only small deviations occur for low sample size and pattern dimension fractions. The random uniform distribution is therefore adopted as a standard distribution for measuring the classifier's capacity.

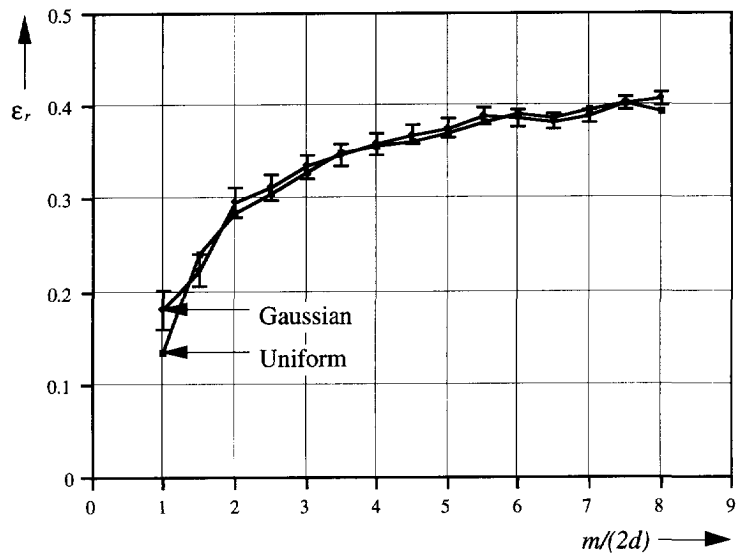

Figure 6·2: Experimental results of the measured average re-substitution error for Fisher's linear discriminant and identical 7-dimensional Gaussian distributions. The uniform (d=7) curve from figure 6·1 is shown for comparison. The error bars represent the 95% confidence intervals using student-t statistics.

When [Cover 1965] analyzed the capacity of the linear classifier, he introduced the restriction that the data points are in *general position* (section 3·6·1). A geometrical restriction is needed because for data points that are *not* in general position the

classifier's capacity is lower. For any classifier that is capable of implementing a non-linear decision boundary, a similar restriction holds. It is often difficult, however, to formulate this in mathematical terms.

An important conclusion is that the classifier's capacity in practice can be limited by a structure that exist in the data. All examples in table 6·1 are high dimensional classification problems ($d>32$) and for high dimensional problems it becomes likely that some kind of structure or correlation exists in the data. In remote sensing, for example [Kamata 1992], a strong correlation in the frequency domain can occur, due to the spatial structure of the data.

The NETtalk experiment that will be discussed in the next chapter is a *189* dimensional problem with a strong correlation in the data. From the *189* features only *7* are high (value one) and the rest are low (value zero). When the *7* high features are known, the remaining features are determined (see section 7·2).

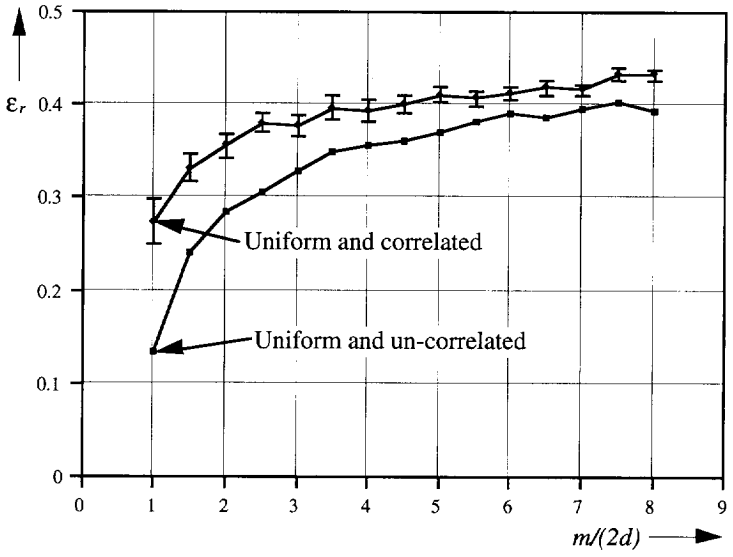

Figure 6·3: Experimental results of the measured average re-substitution error for Fisher's linear discriminant and identical 7-dimensional correlated uniform distributions. The un-correlated uniform (d=7) curve from figure 6·1 is shown for comparison. The error bars represent the 95% confidence intervals using student-t statistics.

In the Foley approach the uniform features are chosen statistically independent from each other, so this assumption influences the classifier's capacity. The NETtalk correlation is introduced here as follows. At random with uniform probability, one feature is selected from the seven-dimensional pattern vector. This feature is set to the value one and the other six features are set to value zero. A data set with two classes is generated at random as before and the re-substitution error is measured for different sample sizes. The results are shown in figure 6·3.

Figure 6·3 clearly illustrates that the capacity of the linear classifier for such a data set is lower, resulting in a graph that is always higher than for the original uniform data. These results suggest that the bias between re-substitution error and true error is smaller for equal sample size.

6·4·2 Applying the Foley approach to feed forward classifiers

The Foley capacity approach can be applied to any classifier with finite Vapnik-Chervonenkis capacity V. It is known that the Vapnik-Chervonenkis capacity is bounded for feed forward classifiers by 6·6. In this section some of the factors that influence the feed forward capacity will be investigated, by using the Foley average-capacity measure.

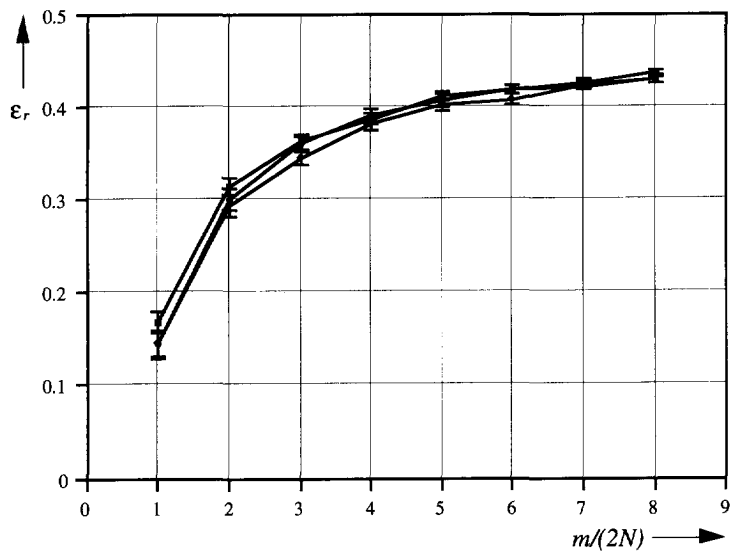

Figure 6·4: Experimental results of the measured average re-substitution error for three feed forward classifiers with 7 hidden units and 7, 14, and 21-dimensional equal uniform distributions. The error bars represent the 95% confidence intervals using student-t statistics.

A systematic analysis of all factors that influence the Foley capacity analysis is almost impossible due to the large number of design issues that are involved. This conclusion was already mentioned in section 5·6·4 but is also valid here. Therefore to limit the number of Monte-Carlo experiments, the conjugate gradient descent method (CG) 4·25 is used. The advantage of this method is that it uses a variable learning rate so neither learning rate η nor a momentum α need to be chosen.

In figures 6·1, 6·2 and 6·3 the re-substitution error was plotted as a function of the fraction $m/(2d)$ but this is not a useful measure for feed forward classifiers. The criticism of statistical pattern recognition focused on the ratio of sample size to

trainable parameters therefore the fraction $m/(2N)$ will be used now. The equal probable, uniform and independent data set (see section 6·4·1) was used again to measure the Foley's capacity fingerprint.

The measurements shown in figure 6·1 showed that ε_r was independent of the pattern dimension. This is investigated in figure 6·4 for the following feed forward networks: *7-7-1*, *14-7-1* and *21-7-1*. The error bars represent the 95% confidence intervals using the student-t statistics. It should be remembered that N is *64*, *113* and *162* for the three different experiments respectively, which means that the sample sizes differ more than a factor two between the curves. The conjugate gradient descent was stopped after *100* iterations, a number that was arbitrarily chosen.

The capacity of a feed forward classifier seems to be almost independent of the data dimension d. The confidence intervals overlap or almost overlap and this result leads to the decision to perform the following experiments only for $d=7$ to save computer simulation time. In section 2·4·4 the small sample size properties were shown to be dependent on the pattern dimension, which is not in contradiction with the conclusion of these experiments. The expected probability of error ε is dependent on the true probability distributions and therefore the dimension d. However, the capacity seems to be independent of the pattern dimension in these simulations.

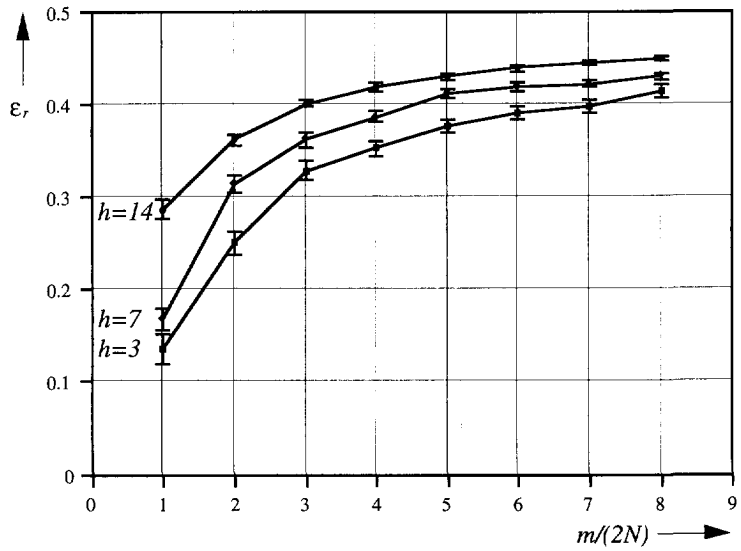

Figure 6·5: Experimental results of the measured average re-substitution error for three feed forward classifiers with 3, 7 and 14 hidden units for equal uniform distributions (d=7). The error bars represent the 95% confidence intervals using student-t statistics.

The universal approximation properties discussed in section 4·2·2 required that enough hidden units be available to approximate the desired function. The theorem on

the maximum number of hidden units ensured that for a finite data set a finite number of hidden units is sufficient. The complexity of the feed forward classifier is clearly determined by the number of hidden units.

In figure 6·5 experimental results are shown for 7-dimensional equal uniform data sets. The number of hidden units is chosen at *3*, *7* and *14* for the different experiments, corresponding to *N=28*, *64* and *127* respectively. The results reveal that the Foley's capacity plotted versus the fraction *m/2N* decreases with increasing number of hidden units. This seems to contradict the earlier statement that the capacity increases with the number of hidden units. This is true only the capacity does not increase linearly with the number of added parameters.

The consequence of this observation is that simple comparison of the number of weights *N* to the number of training samples *m* is too pessimistic for network classifiers with a large number of hidden units. This conclusion is supported by the redundancy that was noticed in section 6·3.

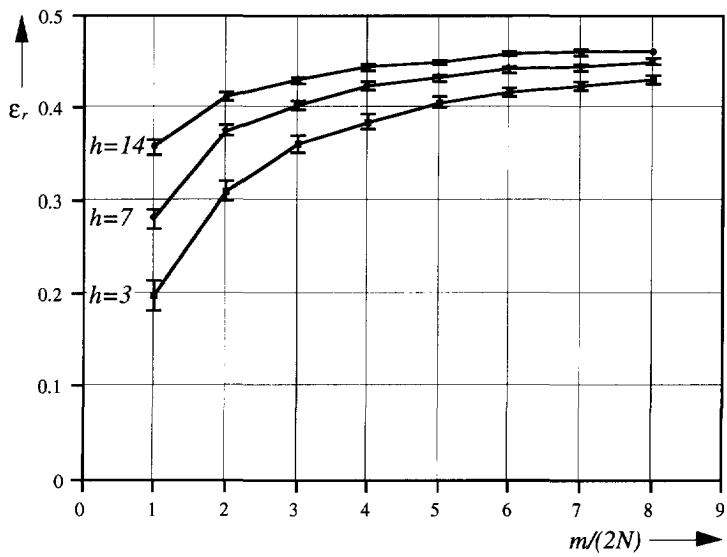

Figure 6·6: Experimental results of the measured average re-substitution error for three, feed-forward classifiers with 3, 7 and 14 hidden units for equal uniform distributions. The conjugate gradient descent was stopped after 50 iterations and the error bars represent the 95% confidence intervals using student-t statistics.

The relation between regularization of equation 5·3 and the early stopping of an iterative training method was discussed in section 5·3·5. A finite number of steps implies that the algorithm will stop in a bounded distance from the starting point and can not therefore exploit the complete parameter space. This conclusion is true for steepest descent with fixed step size but for variable step-size methods this is less obvious.

The experiments of figure 6·5 are repeated, but the number of line minimizations is now reduced to *50* instead of the original *100*. A careful examination of figure 6·6 discloses that the capacity is limited by this early stopping. All graphs of figure 6·5 are shifted upwards in 6·6. A network classifier with more hidden units is in this interpretation less powerful if it is trained with less iterations. Sample size considerations are thus dependent on the effectiveness of the training algorithm used and this fact can not be neglected.

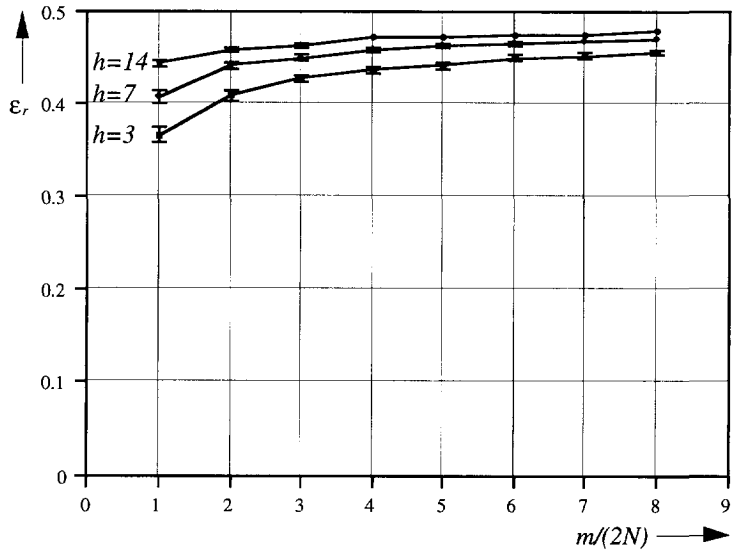

Figure 6·7: Experimental results of the measured average re-substitution error for three feed forward classifiers with 3, 7 and 14 hidden units for equal uniform but correlated *distributions. The conjugate gradient descent was stopped after* 100 *iterations (line minimization's) and the error bars represent the 95% confidence intervals using student-t statistics.*

An important issue of the previous section was the possibility that a special structure in the data can limit the actual capacity. The correlation that was induced for experiment 6·3 is used for a similar experiment using feed forward classifiers. These results are shown in figure 6·7 and indicate a large reduction of the network's capacity for such data. This effect appears to lead to the most dramatic reduction of the feed forward network's capacity. The graph for architecture *7-14-1* in figure 6·6 suggests that a sample size below $m<2N$ can be possible for special data sets.

The existence of structure in data from real world problems is rarely easy to detect. An example of such a problem is analyzed in the next chapter, where the NETtalk problem is discussed. Methods to estimate the *intrinsic dimension* of a data set, as this phenomenon is sometimes called, can for example be found in [Trunk 1976], [Pettis 1979], [Wyse 1980] and [Fukunaga 1971]. These methods unfortunately do not give

any information about the type of correlation that exists in the data. They only return an estimate of the minimum number of parameters needed to describe the data.

6·4·3 Discussion of the Foley capacity experiments

The rule of thumb that the ratio $m/2N$ should be larger than a factor ten is probably a good design criterion to keep in mind, especially if not much knowledge about a certain application is available. The results of this section clearly show that the classifier's capacity is more complicated and dependent on a number of design settings.

The main conclusion from these simulation results is that a simple comparison of the number of weights N to the number of training patterns m is not enough to support the criticism from statistical pattern recognition. The analysis in figure 6·5 shows that the capacity does not increase linearly with the number of added parameters. For applications with a large number of hidden units, the requirement that $m/2N>10$, becomes a pessimistic guideline.

If this last conclusion is combined with an early stopping of the training procedure and a strong correlation or structure in the data set, then an extrapolation of figure 6·7 suggests that low sample size applications can be possible. I would like to emphasize, however, that without detailed knowledge of the experimental settings and information about a possible structure in the data, it is impossible to make a strong statement about the conditioning of an application.

6·5 Sample size and probability of error

6·5·1 Introduction

The expected probability of error of a classifier is, as discussed in section 2·4·4, dependent on the sample size, true underlying distributions, pattern dimension and the parametric form chosen for the classifier. In the previous section the capacity of a feed forward classifier was investigated using Foley's capacity. The consequence of the capacity on the expected probability of error is still an open question and in this section this relation will be investigated.

The influence of the underlying distribution is complex and as far as I know the only theory that exists on generalization is the Vapnik-Chervonenkis theory. This theory completely ignores any distribution, having the advantage that it is distribution free and that it can be applied to any classification problem. The disadvantage of this approach is that it is a very pessimistic bound because a specific distribution can dramatically reduce the number of required samples.

The small sample size properties of feed forward classifiers will experimentally be observed for a number of different data sets and network classifiers. The advantage of this approach is that the results are accurate. The clear disadvantage is that the results

can not directly be generalized to other problems. The experiments reported here are, therefore, not intended to serve as a strict guideline for classifier design, but to increase understanding of the generalization capabilities of feed forward classifiers.

6·5·2 A data set for small sample size experiments

The small sample size properties of the linear classifier or the simple perceptron are *classically* studied for the Gaussian distribution, as seen in section 3·6·2. The theoretical and experimental curves are mostly for the common covariance case where the Bayes optimal classifier is a linear classifier. It seems, therefore, that this distribution is an obvious choice for sample size experiments because of its importance in traditional experiments.

A linear classifier can be implemented by the multi-layer perceptron but the popularity of this new type of classifier is due to its non-linear capabilities. The number of hidden units h controls the classifier's complexity or non-linearity. This aspect should not be ignored in an experimental evaluation. The following data set is therefore proposed and is inspired by equation 5·2:

$$\mathbf{x}_{\omega_i} = \begin{cases} \mathbf{x}_1 = (r_{\omega_i} + \psi)\cos\gamma_1 \sin\gamma_2 \sin\gamma_3 \sin\gamma_4 \sin\gamma_5 \sin\gamma_6 + \zeta_1 \\ \mathbf{x}_2 = (r_{\omega_i} + \psi)\sin\gamma_1 \sin\gamma_2 \sin\gamma_3 \sin\gamma_4 \sin\gamma_5 \sin\gamma_6 + \zeta_2 \\ \mathbf{x}_3 = (r_{\omega_i} + \psi)\cos\gamma_2 \sin\gamma_3 \sin\gamma_4 \sin\gamma_5 \sin\gamma_6 + \zeta_3 \\ \mathbf{x}_4 = (r_{\omega_i} + \psi)\cos\gamma_3 \sin\gamma_4 \sin\gamma_5 \sin\gamma_6 + \zeta_4 \\ \mathbf{x}_5 = (r_{\omega_i} + \psi)\cos\gamma_4 \sin\gamma_5 \sin\gamma_6 + \zeta_5 \\ \mathbf{x}_6 = (r_{\omega_i} + \psi)\cos\gamma_5 \sin\gamma_6 + \zeta_6 \\ \mathbf{x}_7 = (r_{\omega_i} + \psi)\cos\gamma_6 + \zeta_7 \end{cases} \tag{6·7}$$

The data set contains two equally probable classes that are distinguished by the class dependent parameter:

$$r_{\omega_i} = \begin{cases} 6.2 & \mathbf{x} \in \omega_1 \\ 10 & \mathbf{x} \in \omega_2 \end{cases} \tag{6·8}$$

The statistics of ψ, γ_i and ζ_i are varied to create four different data sets from equations 6·7, that thereby all belong to the same family. The random variables, ψ and γ_i, are described by uniform distributions and ζ_i is a Gaussian distributed variable. If γ_1 to γ_5 are chosen to be zero, then features six and seven become equal to distribution 5·2 (see also figure 5·2). The distribution proposed here is a high dimensional equivalent of 5·2.

The different data sets that are generated from equation 6·7 are listed in table 6·3. In this table N(0,1) specifies a Gaussian distribution with zero mean and unit standard deviation. Furthermore the performances of the following two classifiers are listed:

$$g_1(\mathbf{x}) = \mathbf{x}_7 - \mathbf{w}_8 \tag{6·9}$$

and

$$g_q(\mathbf{x}) = \sum_{i=1}^{7} \mathbf{x}_i^2 - 65.61 \qquad (6\cdot10)$$

The performances of these two classifiers indicate the complexity of the data that was varied to create a set of experiments that are representative for a non-linear classifier. The two classifiers 6·9 and 6·10 are the *best* linear and the *best* quadratic classifiers for this distribution. Parameter $\mathbf{w}_8$ is dependent on the statistics of γ_i and was experimentally determined.

Table 6·3: The statistics of the random variables from equation 6·7, for the four data sets that are used. The performances of two traditional classifiers, measured using 20,000 samples, are listed for comparison.

Data set	ψ Uniform	γ_i Uniform	ζ_i Gaussian	g_l $\hat{\varepsilon}_t$	g_q $\hat{\varepsilon}_t$
D_1	0	0	N(0, 1)	0.029	0.045
D_2	0	<-π/3, π/3>	N(0, 1)	0.165	0.043
D_3	0	<-2π/3, 2π/3>	N(0, 1)	0.330	0.041
D_4	<-1.8, 1.8>	<-π/3, π/3>	N(0, 0.22)	0.171	0.004

The statistics are chosen such that data set D_1 is the traditional Gaussian distribution with different mean values for each class but equal covariance structure. The linear classifier is optimal in this case and the quadratic classifier shows a higher error. Data set D_2 is the high dimensional equivalent of distribution 5·2. The difference between the performances of the quadratic and linear classifier becomes significant.

For data set D_3 the difference between the performance of the linear and quadratic classifier is increased even more. The overlap between the classes is reduced in data set D_4 but the *amount of randomness* and the classification complexity is kept equal to D_2. The performance of the linear classifier is almost the same for D_2 whereas the quadratic classifier is significant better.

6·5·3 Experiments and results

The average generalization errors (ε_t) are measured for the three network architectures that were analyzed in section 6·4·2: *7-3-1*, *7-7-1* and *7-14-1*. The conjugate gradient descent method was stopped after 100 line minimizations so the capacity curves of figure 6·5 apply to the experimental settings used here. The expected generalization error is measured using 20 independent training sets and the resulting classifiers are tested with an independent test set containing 2000 samples.

The results of these experiments are plotted in figures 6·8 to 6·11. The generalization errors are plotted as function of the fraction *m/(2N)* and the error bars represent the 95% confidence intervals. The three curves in one graph should be interpreted with care because the number of trainable parameters *N* changes with the architecture. The curves

are thus calculated with a different number of training samples. This explains why the confidence intervals for the smaller networks are larger in most figures.

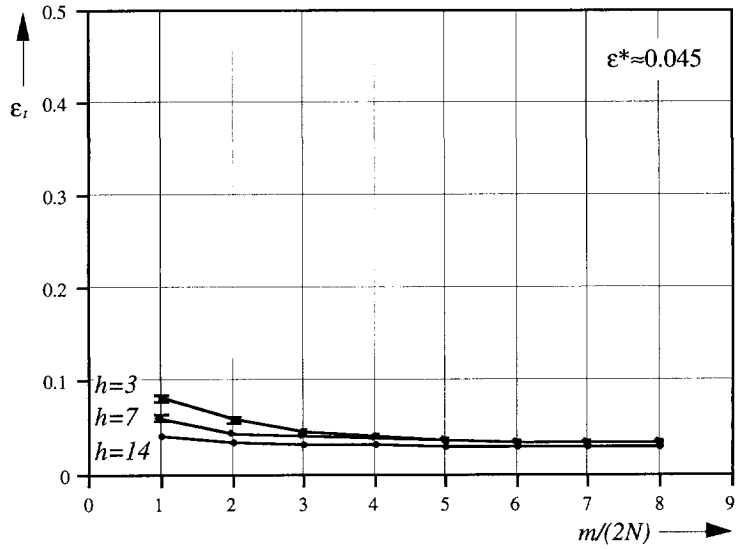

Figure 6·8: The average probability of error of three feed forward classifiers for different training sample sizes. The training sample sizes are varied for different fractions of m/(2N) and the statistics of the data set are given by D_1 in table 6·3. The error bars represent the 95% confidence intervals using student-t statistics.

The experiments in figure 6·8 show a rapid convergence of the average classification error to the Bayes optimal solution. This is not a surprising result because the data consists of two compact clusters and the network model contains the linear classifier, that is optimal for this type of distribution.

The results of the more complex distribution of D_2 in figure 6·9 confirms an earlier statement (section 2·4·4) that small sample behavior is dependent on the data distribution. The optimal decision boundary must now be approximated by sigmoid boundaries and the architecture with *h=3* hidden units, seems to be *not* complex enough to perform this task. Furthermore the generalization errors are significantly larger for small sample sizes.

This last conclusion becomes even more apparent for data set D_3 (figure 6·10) where the data complexity is again increased. The errors induced by the finiteness of the data are much larger and even the classifier with *h=14* hidden units is not complex enough.

The last data set (D_4) is clearly the most difficult application to solve. The optimal decision boundary must be determined precisely and a small deviation can cause a large change in expected performance. In figure 6·11 large confidence intervals are probably caused by this narrow margin and the results of this figure suggest that a much larger sample size is needed.

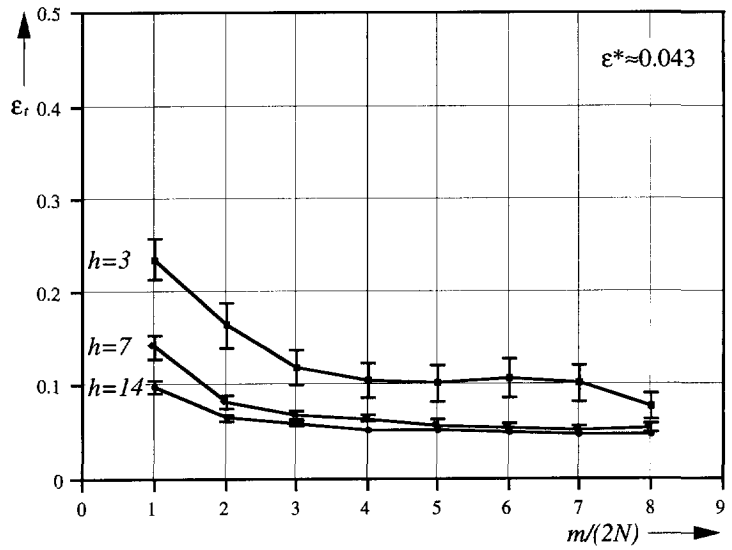

Figure 6·9: The average probability of error of three feed forward classifiers for different training sample sizes. The training sample sizes are varied for different fractions of m/(2N) and the statistics of the data set are given by D_2 in table 6·3. The error bars represent the 95% confidence intervals using student-t statistics.

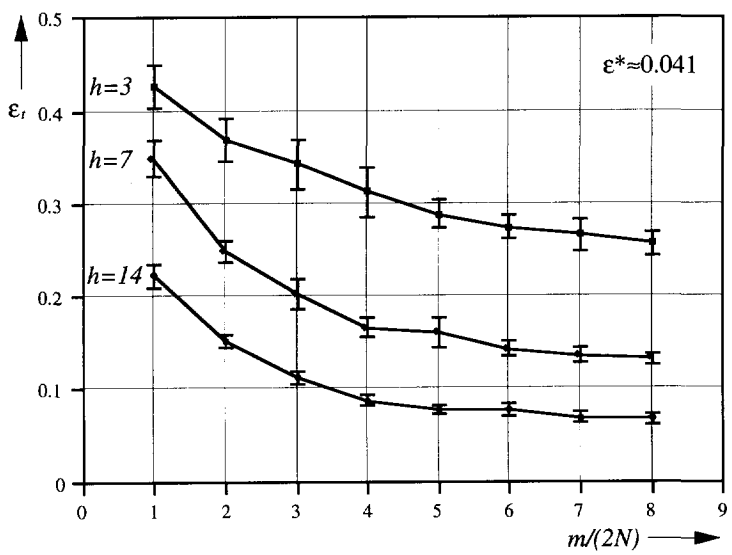

Figure 6·10: The average probability of error of three feed forward classifiers for different training sample sizes. The training sample sizes are varied for different fractions of m/(2N) and the statistics of the data set are given by D_3 in table 6·3. The error bars represent the 95% confidence intervals using student-t statistics.

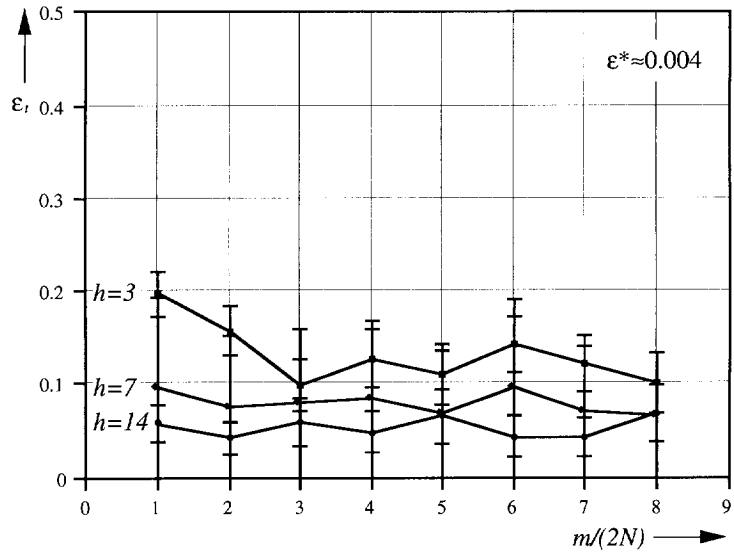

Figure 6·11: The average probability of error of three feed forward classifiers for different training sample sizes. The training sample sizes are varied for different fractions of m/(2N) and the statistics of the data set are given by D_4 in table 6·3. The error bars represent the 95% confidence intervals using student-t statistics.

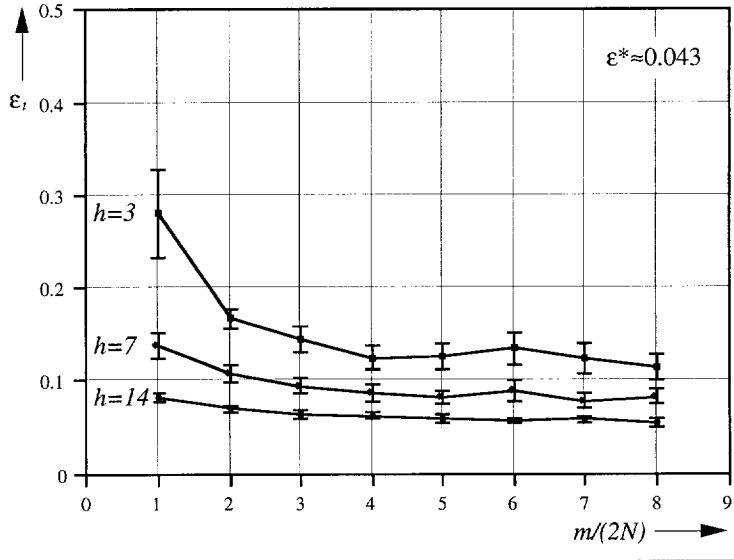

Figure 6·12: The average probability of error of three feed forward classifiers for different training sample sizes. These experiments are equal to figure 6·9 except that the conjugate gradient descent method was stopped after 50 iterations. The error bars represent the 95% confidence intervals using student-t statistics.

A general conclusion that can be drawn from all these experiments is that the ranking of these curves agrees with the capacity curves of figure 6·5. A small re-substitution bias suggests a small error due to finite sample sizes and this hypothesis holds for all four figures that are shown.

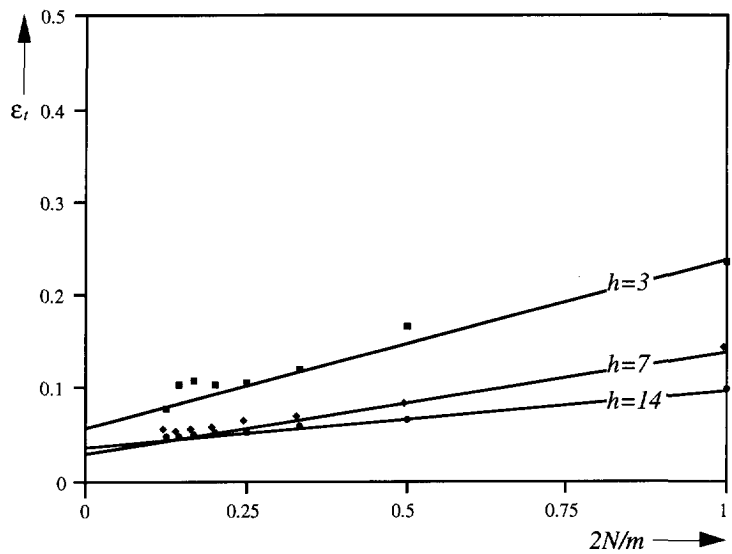

Figure 6·13: A scatter plot, together with a first-order regression, of the average probability of error as function of 2N/m. The measurements are the experiments already plotted in figure 6·9.

The final experiments are plotted in figure 6·12 where a network classifier is trained for recognizing data set D_2. Here the conjugate gradient descent is stopped after 50 iterations so the capacity curves of figure 6·6 apply to these results. The conclusion, that the capacity of network classifiers is limited by an early stopping, is confirmed by a comparison of figures 6·9 and 6·12. A feed forward classifier, with 7 hidden units, is now not complex enough to approach the Bayes optimal solution.

Furthermore the classifier with 14 hidden units in figure 6·12 performs better for low sample size fractions ($m \approx 2N$) than the same classifier of figure 6·9. This observation is closely related to a similar observation in classical pattern recognition by [Raudys 1976]. In section 2·4·7 it was discussed that if the sample size is low, a linear classifier can perform better than a quadratic classifier even when the quadratic classifier is Bayes optimal. The selection of the *best* network classifier should also be based on criterion 2·18 as suggested by [Raudys 1976].

6·5·4 A first-order approximation to small sample size effects

An asymptotic expansion of the expected probability of error as function of sample size, only exists for the linear classifier applied to Gaussian distributions with equal

covariance structure (see section 2·4·5). These approximations are mostly first-order Taylor series and the expression by [Smith 1972] (equation 3·59), is a good example of such an approximation. Almost all asymptotic approximations are good up to order m^{-1}, which suggests that higher order terms are of much less importance.

A scatter plot of the experiments from figure 6·9 is shown in figure 6·13 where $2N/m$ is plotted against ε_t. In this figure a roughly linear correlation is observed, suggesting that indeed the expected probability of error is proportional to m^{-1}. The observed correlations for all the experiments are listed in table 6·4; the corresponding scatter plots are omitted. The correlations found for experiment 6·11 are listed for completeness. This data is unreliable, however, due to the large variance found in the experiments (see figure 6·11). This observation is therefore ignored and from table 6·4 it can be concluded that the suggested proportionality is a more universal law. The properties of data set D_4 are interesting for further research.

Table 6·4: The observed correlation, between $2N/m$ and the expected probability of error, for the experiments described in section 6·5·3.

	$\hat{\rho}$		
Figure	*h=3*	*h=7*	*h=14*
6·8	0.993	0.997	0.998
6·9	0.985	0.994	0.995
6·10	0.954	0.992	0.992
6·11	0.848	0.564	0.025
6·12	0.984	0.987	0.984

The observed proportionality between m^{-1} and the expected probability of error is supported by observations from [Kraaijveld 1993]. Theoretical support can be found in [Amari 1992b] and [Kang 1993] although some assumptions that are made to arrive at this theoretical proportionality are doubtful according to the authors.

6·5·5 Correlation between capacity and generalization

Without explicitly stating it, the Foley capacity analysis of linear or feed-forward classifiers, assumes that some kind of correlation exists between curves found in figures 6·5 to 6·7 and the corresponding curves for the expected probability of error. The rule-of-thumb of section 2·4·6 is based on the analysis by [Foley 1972] and supposes that when the capacity curves start to level-off, the expected probability of error is starting to level-off too.

An experimental demonstration of this correlation can be obtained by the following two approaches. First the correlation between the average re-substitution error of Foley and one of the experimental curves of section 6·5·3 can be directly measured. The second approach uses the result of the previous section that the expected probability of error is proportional to m^{-1}. If a correlation exists between the capacity and generalization curve, the capacity curve must be proportional to m^{-1} as well.

One scatter plot of the Foley re-substitution error against the average probability of error is shown in figure 6·14. The measured generalization errors are the experiments already found in figure 6·9. In this figure a linear correlation is observed, which justifies the followed analysis of section 6·4.

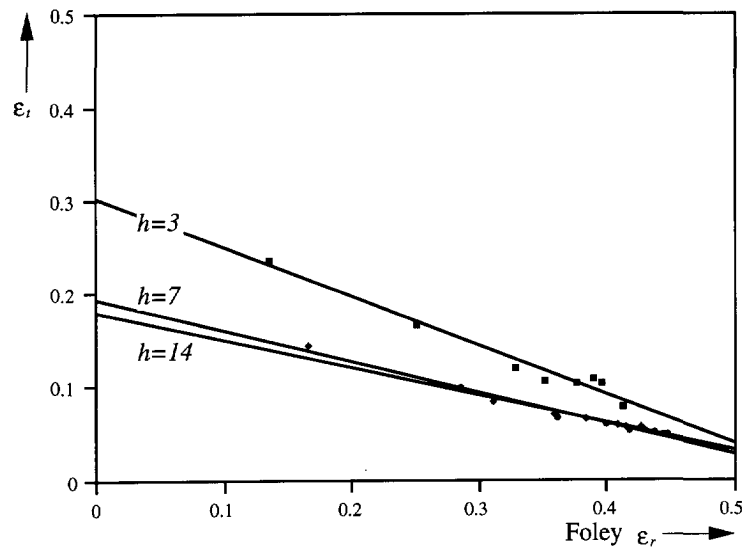

Figure 6·14: A scatter plot, together with a first-order regression, of the Foley re-substitution error against the measured generalization error for the experiments shown in figure 6·9.

The measured correlations for all experiments of section 6·5·3 are listed in table 6·5 and the corresponding scatter plots are omitted. These results reveal a strong correlation between the re-substitution capacity measure and measured generalization errors. The correlations found for experiment 6·11 do not fit very well. This data, however, is very unreliable due to the large variance found in the experiments (see figure 6·11). This observation is again ignored and it is concluded that a strong correlation exists between the capacity as defined in section 6·4 and small sample size properties.

Table 6·5: The observed correlation, between Foley's re-substitution error and the probability of error, for the experiments described in section 6·5·3.

	$\hat{\rho}$		
Figure	*h=3*	*h=7*	*h=14*
6·8	-0.995	-0.997	-0.997
6·9	-0.986	-0.991	-0.979
6·10	-0.980	-0.993	-0.998
6·11	-0.845	-0.565	+0.013
6·12	-0.963	-0.983	-0.982

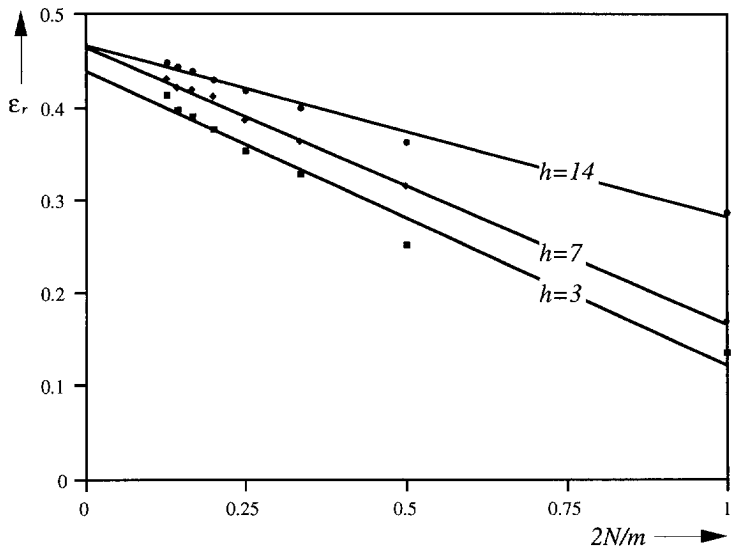

Figure 6·15: A scatter plot together with a first order regression of the Foley re-substitution error as function of 2N/m. These curves correspond to the experimental results of figure 6·5.

It was argued that if a correlation exists between the Foley's re-substitution errors and the generalization errors then the capacity curves should be proportional to m^{-1}. In figure 6·15 a first-order regression is shown between $2N/m$ and the measured re-substitution errors of figure 6·5. This plot indeed reveals that such a relationship exists and in table 6·6 the measured correlations are listed for all three experiments of section 6·4·2.

Table 6·6: The observed correlation between $2N/m$ and Foley's re-substitution error for the experiments described in section 6·5·2

	$\hat{\rho}$		
Figure	*h=3*	*h=7*	*h=14*
6·5	-0.989	-0.999	-0.994
6·6	-0.995	-0.999	-0.996
6·7	-0.949	-0.939	-0.979

The experimental results of this section confirm that a strong relationship exists between the results of the capacity analysis of section 6·4 and small sample size behavior of feed forward classifiers. An interesting observation is that Foley's re-substitution estimate is proportional to m^{-1} and that the observed generalization is also proportional to m^{-1}.

This last result or the first-order approximation found in the previous section, initiates further research. It is interesting to investigate the possibility to predict the future

performance of a feed forward classifier by extrapolating a scatter plot similar to 6·13 or 6·14 for a new application. If for example two or three points of the curve are known, the expected performance at a different working point can probably be predicted. This idea is close to *bootstrap* techniques for error estimation [Jain 1987] that are known in pattern recognition.

6·5·6 Discussion of the small sample size experiments

The experiments described in this section have shown that the analysis of feed forward classifiers using Foley's approach (section 6·4) is valuable to predict small sample behavior of these systems. All results show that the ranking of the measured generalization curves can be explained from the equivalent capacity curves. The conclusion from the Foley experiments, that for networks with a large number of hidden units the requirement $m/2N>10$ is a pessimistic (conservative) guideline, is thereby confirmed.

The conclusion that an early stopping of the training algorithm limits the classifier's capacity is clearly illustrated by these experiments. It was also observed that in small sample size situations a limitation of the training algorithm can increase generalization properties. This is observed in traditional pattern recognition and is known as the peaking effect.

A first-order approximation indicates that generalization is proportional to m^{-1}. It is furthermore interesting that all capacity curves show the same proportionality and this relation is probably an interesting starting point for further research. This research might indicate that a handbook with capacity curves for all known classifiers can be valuable for pattern recognition practitioners.

6·6 Discussion

The small sample size properties of feed forward classifiers have been investigated in this chapter. It was noticed that a number of successful applications can be found in literature that show remarkable results from a traditional point of view (table 6·1). A bound on poor generalization developed by Vapnik-Chervonenkis can not explain any of these successes and this theory seems to support the traditional criticism.

In [Kraaijveld 1993] the Vapnik-Chervonenkis theory is refined and the capacity V is experimentally investigated. It was observed in this reference that in practice the capacity is much lower and thereby the utility of the applications of table 6·1 becomes plausible.

A large redundancy in weight space was experimentally shown for three low dimensional distributions. These experiments hint that the weights in the output layer are significantly more important and that sample size considerations should focus on these weights.

This idea is supported by an analysis of the capacity of feed forward classifiers that is inspired by the [Foley 1972] approach. It was shown that the capacity of a network classifier does not increase linearly with the total number of parameters if hidden units are added. A simple comparison of the number of trainable parameters to the sample size is therefore too pessimistic for classifiers with a large number of hidden units.

The evaluation of the generalization properties for a number of artificial distributions confirms the conclusions from the capacity analysis. A strong correlation exists between the generalization properties and the capacity analysis except for one data set which was ignored because of its large variance. The properties of this data set are interesting for further research.

The generalization capabilities are roughly proportional to m^{-1} and it is furthermore interesting that all capacity curves show the same proportionality. This relation is probably also an interesting starting point for further research.

7
Experiments with Network Classifiers

7·1 Introduction

In the introduction of this thesis it was discussed that when no knowledge is available about the underlying probabilities, no preference can be given to any proposed classification model. This statement is true if for all alternative classification models the asymptotic properties are known to be Bayes optimal and enough data is available. The infinite sample size properties of classical pattern recognition methods can be found in standard text books on pattern recognition ([Duda 1973]).

For the feed forward classifier the asymptotic properties were discussed in section 4·4·2 and this classifier is Bayes optimal if it is complex enough to represent the classification boundary. The non-linearity found at the output of these classifiers alters the pattern error function such that the classifier approximates minimum error performance, a favorable property if the classifier's complexity is lower than the data complexity.

If a finite data set is available then a classifier with only a few adjustable parameters or low capacity is preferred. This preference is important to avoid the *peaking* problem but has shown to be a complex issue in neural classifiers (chapter 6). The criticism of pattern recognition researchers concerning the large number of free parameters is not fully supported by the conclusions of the previous chapter but is still a topic for research.

The small sample properties are also dependent on the underlying distributions. This was already known in statistical pattern recognition (section 2·4·4) and was observed for feed forward classifiers in the previous chapter. In applications the data set is mostly finite and the conclusion that can be drawn from these two facts, the dependency on sample size and probability distribution, is that the applicability of this new family of classifiers should also be investigated on *real-world* problems.

In the neural network literature a large number of publications describe applications of feed-forward classifiers to all kind of real-world problems [Hertz 1991, page 130]. An open question is how neural classifiers compare to traditional pattern recognition methods. A weak point of most publications on neural network applications is that the performance is only reported for different feed forward architectures.

This chapter is meant as a first attempt to compare neural classifiers with some simple pattern recognition methods for real-world problems. In this chapter the

NETtalk experiment, an important success of neural networks, is investigated from a pattern recognition point of view. In the next section some results are shown for three well-known sets namely 80X, IMOX and Fisher's Iris data. Some recent public accessible sets are evaluated in section 7·4 and this chapter ends with a discussion of these results.

7·2 The NETtalk experiments

One of the successful applications of feed forward classifiers that is often discussed in text books on neural networks (see [Hertz 1991] or [Wasserman 1989]), is the NETtalk experiment described in [Sejnowski 1986] and [Sejnowski 1987]. In these two papers an experiment is described where a network classifier learns to convert English text to speech.

Reading aloud English text is considered to be difficult because most phonetic rules, for translating letters to speech are context sensitive. In addition to the fact that this application is complicated to solve, it was also one of the examples listed in table 6·1 that are interesting from a traditional point of view.

The work that is reported in this section is the result of a cooperation between David Levelt and me. The following presentation only covers the main results and many details and results are skipped here. A number of omissions and contradictions that were found in the two papers are omitted here because the main results easily get lost in that discussion. A complete description of this work can be found in [Levelt 1993].

7·2·1 A description of NETtalk

English pronunciation has been extensively studied by linguists and much is known about the correspondence between letters and the elementary speech sounds of English that are called *phonemes* [Venezky 1970]. In most dictionaries, see [Webster 1973] for example, every word is directly followed by a list of phonemes that describes the pronunciation. The problem of reconciling rules and exceptions of how words map to phonemes has traditionally been approached with rule-based knowledge systems [Haas 1970].

An alternative approach is suggested by [Sejnowski 1986, 1987]. Here a feed forward network is used to perform the text to phoneme mapping. The network is trained with a set of commonly used words in English and is tested on a medium sized dictionary. The interesting point is that Sejnowski reformulated this mapping problem such that statistical pattern recognition methods can be applied. This reformulation in statistical terms is not new and can also be found in [Lucassen 1984].

The network and training algorithm

The network that was used is the standard feed forward architecture as shown in figure 4·1 and given by equation 4·2. The non-linear function that was used is given by

3·49. In figure 7·1 a schematic arrangement of the system is shown. Text is fed to inputs of the network and the outputs are trained to predict the phoneme of the central letter applied to the network.

The encoded letters before and after the central letter, provide a partial context for the decision. The size of this window, seven symbols, is inspired by the results published in [Lucassen 1984]. A word is stepped through the window such that every letter is once the central letter to be classified. Every letter therefore generates one phoneme. It is not true that all words generate the same number of phonemes as letters and to handle this problem, a silent phoneme is introduced.

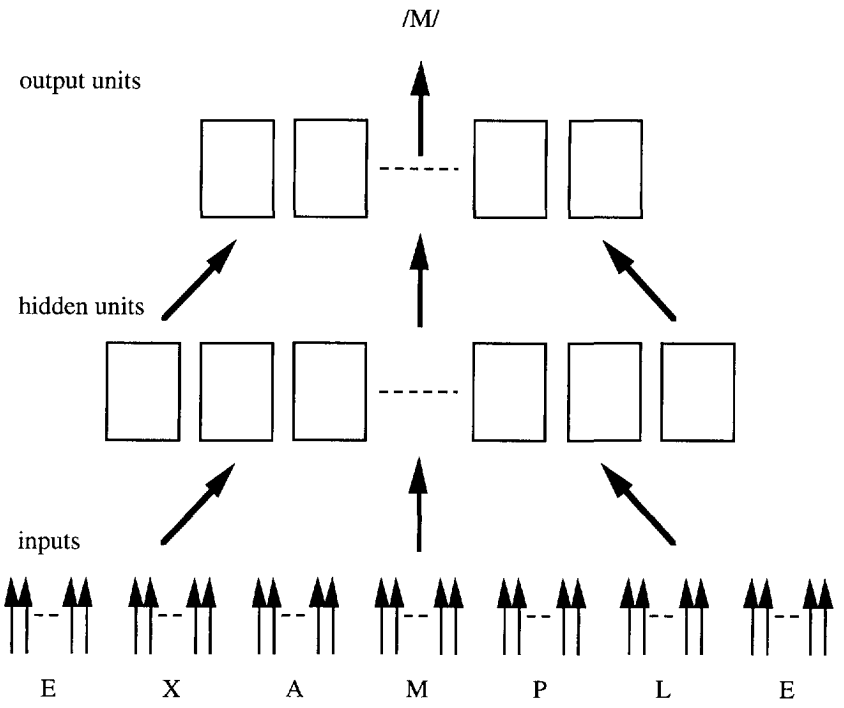

Figure 7·1: A schematic drawing of the network setup. The input units at the bottom encode the text. One layer of hidden units, receive the inputs and one output layer is used to encode the phonemes. An example *input string is shown together with correct output phoneme /M/.*

The reported networks are trained using a modified back-propagation algorithm (equation 4·13). The first modification is that the error is propagated only if it is larger than a margin of 0.1, to ensure that the network is not over-trained. This can be regarded as some form of regularization (section 5·3·5) and is an interesting detail for the discussion in section 6·1.

The sample-update version of 4·13 is just an approximation of the true steepest descent given by equation 4·10. In NETtalk neither 4·10 nor 4·13 is used but the

gradient is accumulated for each word, which on average means that the gradient is estimated with 5 samples. The initial weights of the network were chosen as random numbers uniformly distributed between -0.3 and 0.3.

Representation of letters

In the experiments described by [Sejnowski 1986, 1987] seven groups of inputs encode the input text (see figure 7·1). The letters are represented in each group by 26 inputs for each letter of the alphabet, plus 3 additional units to encode punctuation and word boundaries. Within a group only *one* input is high and the other inputs in that group are low. The input that is high specifies the letter, punctuation or word boundary that is represented by this group of 29 inputs. This leads to a total of 7x29=203 inputs of which exactly 7 are high (value 1) and the remaining inputs are low (value 0). The input data, although processed as real valued, is in fact binary valued and highly correlated.

Representation of phonemes and performance estimation

The 52 phonemes that are commonly distinguished in English are encoded with 23 *articulatory* features such as point of articulation, voicing and vowel height. Phonemes are formed by three to five articulatory features. A unique relation exists between phonemes and the corresponding features (see [Sejnowski 1986, Table 1]). The reverse mapping is *not* unique because only specific combinations of articulatory features correspond to phonemes. The 23 features are directly encoded by 23 units and three additional units are used to encode stress and syllable boundaries. This leads to a total of 26 output units, which is a *distributed* representation of the 52 phonetic classes.

Two measures of performance are introduced. A sample is considered a *perfect-match* if the value of *each* articulatory feature is within a margin of 0.1 of its correct value. The *best-guess* criterion considers a sample correct, if the output vector makes the smallest angle with the corresponding phoneme. All phonemes can be considered as prototypes in the articulatory feature space. The network output is a vector in this space and the angle of this vector to all the 52 prototypes is calculated. The network output is *classified* to the phoneme with the smallest corresponding angle.

The data sets

In both papers a number of experiments are discussed and not all of these are interesting from a pattern recognition point of view. In the next sections only the *dictionary experiments* from [Sejnowski 1986, 1987] will be investigated. A 20,012 word corpus was taken from Merriam Webster's pocket dictionary. From this set the 1000 most common words in English were selected as training data.

This choice of training and testing data has two implications from a statistical point of view. First the test data is not completely independent because the training data is part of it. The training data is only 5% of the complete set so the bias can be assumed to be small. Furthermore this estimate is not representative for the classifier in practice because the relative frequency of occurrence in normal text is ignored. The practical

performance will be somewhere between the re-substitution and test set error, presumably closer to the re-substitution error than the test set error.

A second issue is that the training set is not randomly chosen from the word corpus. Most exceptions in English are found in the commonly used words so this training set is surely not representative for the statistics of complete population.

Experiments and conclusions

The results reported in [Sejnowski 1986, 1987] are difficult to interpret. After a close reading, one must conclude that most performances are re-substitution errors. Despite the overlap between training and testing set, the results are the performances measured on the 1000 most commonly used words or the complete test set is used for a few extra training sweeps.

The *perfect-match* performances (see paragraphs on performance estimation in this section) reported in [Sejnowski 1986, 1987] are not interesting, in my opinion, because all reported classification results are worse than $\hat{\varepsilon}_r > 0.52$. The perfect match is therefore not fully investigated in the following comparison .

The interesting results are obtained with the *best-guess* measure. A network with no hidden units (*203-0-26*) which is a set of simple non-linear perceptrons, can accomplish a re-substitution error of $\hat{\varepsilon}_r = 0.12$. The performance increases with the increase of hidden units to $\hat{\varepsilon}_r = 0.02$ for 120 hidden units (*203-120-26*). The test set error for this classifier is reported to be $\hat{\varepsilon}_t = 0.23$. This network is trained a few extra sweeps with 5 passes using the dictionary set and the performance improves to $\hat{\varepsilon}_r = 0.10$.

It is further noted that best-guess errors are far from random. Most confusions were between phonemes that were very similar. This observation can be understood if one realizes that the errors made by the network are errors made in the articulatory feature space. Any inaccuracy in this space is likely to result in a confusion of phonemes that are close in sound. A similar idea is sometimes used in binary coding and is then called a *gray-code*.

The authors [Sejnowski 1986, 1987] conclude that the network was capable of reaching a significant level of performance using a neural network with a window of only seven letters. One can argue that most of their results are re-substitution errors. The last result of $\hat{\varepsilon}_r = 0.10$, however, is measured on a set of 20,000 words which can be considered large. Furthermore, the relative frequencies of occurrence are neglected together with the fact that most errors are close in sound. The system is probably sufficiently accurate for simple, telephone quality applications.

7·2·2 A duplication of NETtalk

We have repeated the experiments by [Sejnowski 1986, 1987] and the simulations and results are reported in this section. The data sets used for the original NETtalk were unfortunately deleted after the publication. With the help of the authors, reconstruction of the training and testing data have been made and became publically accessible as part of a neural network benchmark (for details see [Levelt 1993] and appendix A).

These sets were used in the following experiments, although it showed some inconsistencies with the description found in the original paper.

It appeared that one articulatory feature was missing in the reconstructed set and that the stress and syllable boundaries were coded differently (see [Levelt 1993]). This reduced the number of outputs to 24 instead of the 26 used in the original experiments. It was furthermore not clear if the stress and syllable codes were used in the evaluation of the network performance because this information was used to enhance the speech generation. In the experiments of this sub-section this information is ignored and 22 outputs are used.

The letters that are encoded by the inputs required three additional units per letter to encode punctuation's and word boundaries in [Sejnowski 1986, 1987]. For the dictionary experiments only word boundaries (spaces) are needed. The two other units are always zero for these experiments and were therefore removed, leading to 27 inputs per letter. Together with the previous change, the network architecture that we have used, reduced to *189-h-22*.

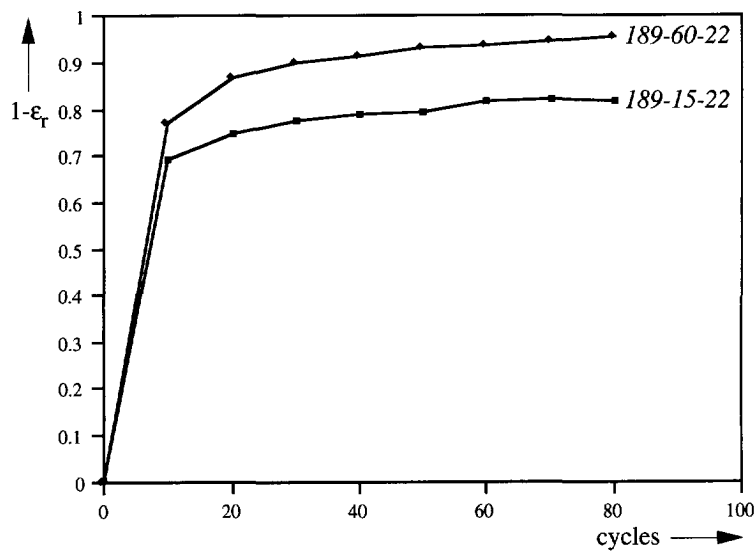

Figure 7·2: The learning curves for training two types of feed forward classifiers using the 1000 most commonly used English words. The performance that is shown is the fraction of correctly classified samples on the training set, using the best-guess measure.

The simulations were performed on a SUN SparcStation using the software library described in chapter 8. The back-propagation method was modified according to the changes described in the section 7·2·1. The learning rate and momentum, were chosen η=0.1 and α=0.9 respectively, that is equivalent to the values published in [Sejnowski 1986, 1987]. The initial weights of the network were chosen as random numbers uniformly distributed between -0.3 and 0.3.

In figure 7·2 the learning curves are shown for networks with 15 and 60 hidden units. The percentage of correctly classified phonemes on the training set, is plotted as a function of the number of training cycles. This figure can directly be compared with figure 7 in [Sejnowski 1986]. The best-guess measure is shown and the perfect match is omitted in table 7·1 because it measured an error of 1 for all experiments that were performed. The best-guess results are comparable to the results published in [Sejnowski 1986, 1987], except that we needed 80 cycles against 30 cycles in the original publication. Furthermore the network with 120 hidden units did not learn in a stable way.

Table 7·1: A comparison of the NETtalk experiments reported by [Sejnowski 1986] and our results. These results are based on the best-guess measure.

	[Sejnowski 1986]		Duplication	(Levelt&Schmidt)
Architecture	$\hat{\varepsilon}_r$	$\hat{\varepsilon}_t$	$\hat{\varepsilon}_r$	$\hat{\varepsilon}_t$
189-15-22	0.12	not reported	0.182	0.307
189-60-22	0.05	not reported	0.044	0.241
189-120-22	0.02	0.23	unstable	unstable

The final classifiers that were obtained are listed in table 7·1, together with the equivalent results reported by Sejnowski. From these results it can be concluded that the performances claimed in the original paper can be reproduced although not all the experiments can exactly be repeated. The network with 60 hidden-units shows a generalization error that is comparable to the 120 hidden-units networks. The number of training cycles needed to reach this solution was much more than the number reported by Sejnowski. The perfect-match performances, although it was argued that the results are not interesting due to the high errors, could not be reproduced either.

7·2·3 Additional experiments with NETtalk

Two additional experiments were performed with the NETtalk data set. In chapter 5 the sensitivity of the training procedure to the initial weights was examined and it was concluded that this sensitivity plays an important role in the applicability of neural classifiers. The experimental results in [Sejnowski 1986, 1987] are presumably based on one observation.

To investigate the sensitivity for NETtalk a number of networks were trained that only differed in initial weights. The learning rate and momentum were changed to η=0.3 and α=0.0 to be sure that every initialization converged properly. To limit the excessive amount of CPU seconds needed for this experiment the number of hidden units was set to 15. The sensitivity for the initial weights is expected to increase if the number of hidden units is increased, so these results serve as a lower bound.

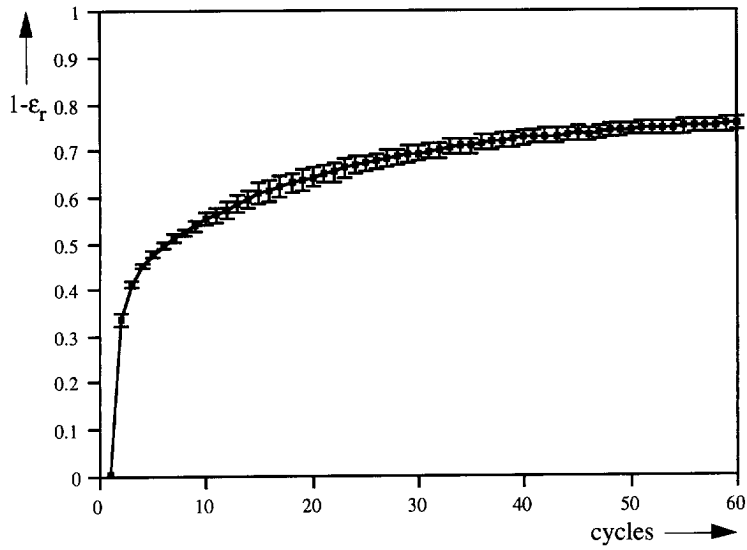

Figure 7·3: The average best-guess performance together with the 95% confidence intervals, measured from 9 networks that differed only in initial weights.

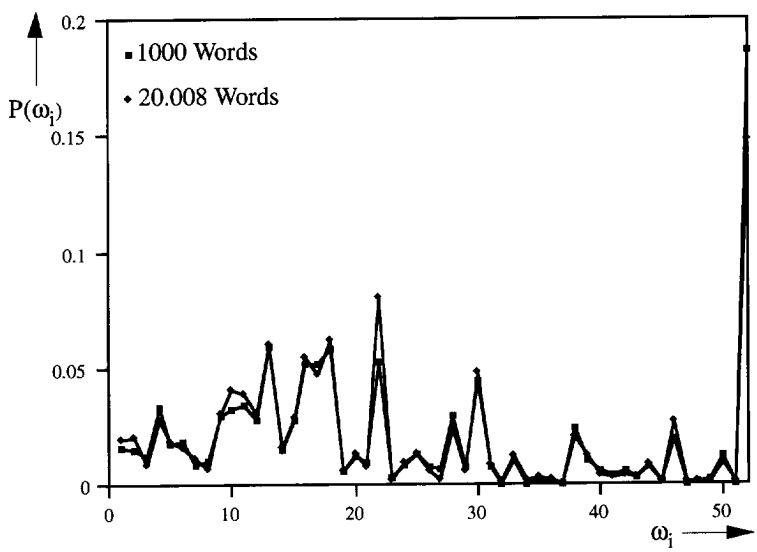

Figure 7·4: The histogram of the 52 phonetic classes for the training and test data set. Phoneme class 52 represents the silent *phoneme.*

The results are plotted in figure 7·3 as a function of the number of training cycles. The difference between the final networks showed a standard deviation of 0.019 on the training set and 0.013 on the test set of 20,000 words. If the results of Sejnowski for 60

and 120 hidden units are compared then it can be concluded that these performances are so close that they could easily have been caused by different weight initializations.

The training set for the NETtalk classifier is not a random set but was deliberately chosen as the 1000 most common words. In statistical pattern recognition it is assumed that a random subset of the universe is chosen. Linguists indicate that most exceptions in English are found in the most commonly used words so this set can not be regarded as a random sampling of the dictionary set. In figure 7·4 a histogram for the training and testing set is shown for the a-priori class probabilities. The statistics of both sets show large similarity and it can be concluded that the classes are properly sampled.

A training set should not only reflect the statistics of the a-priori class distribution but also the statistics of the feature space. The histogram of this 189 dimensional space can not be plotted so a different approach is chosen. A random training set of 1000 words is selected from the dictionary set and is used to train a network classifier with 60 hidden units. The training parameters and initial weights were chosen to be the same as those for the experiments of section 7·2·2.

Table 7·2: The measured classification error for a network with 60 hidden units for two different training sets.

	Classification error	
Training set	$\hat{\varepsilon}_r$	$\hat{\varepsilon}_t$
1000 common words	0.044	0.241
1000 random words	0.024	0.219

The results are listed in table 7·2 and show statistical equivalence, given the variance opserved in figure 7·3. The influence of the initial weight selection is large enough to explain the difference in these two observations. It can be concluded that although Sejnowski did not use a random training set, this is not an important issue.

7·2·4 Metrics in feature space of NETtalk

The question that arises now is how do traditional pattern recognition methods solve the text to phoneme mapping. Another interesting and related question is whether structure exists in the data. Non-parametric pattern recognition methods assume some compactness or clustering of the classes. This compactness is dependent on a chosen measure of distance in feature space. The Euclidean distance measure is an obvious choice but for binary data this reduces to the so called *Hamming* distance. In formula the Hamming distance between two binary patterns $\mathbf{x}$ and $\mathbf{x}'$ is given by:

$$e_H(\mathbf{x},\mathbf{x}') = \sum_i |\mathbf{x}_i - \mathbf{x}'_i| \qquad (7\cdot1)$$

The Hamming distance between two binary values is simply the number of bits that are different. In section 7·2·1 the input representation of NETtalk was explained. This data is binary and a correlation between the bits exists. Exactly 7 bits of the total of 189

are high (value 1) and the rest is low (value 0). As a result of this correlation, the number of distances that can be distinguished in this space using the Hamming metric is only eight (zero to seven).

In this section an alternative distance measure will be used that is meant to improve the classification performance. *A-priori* information is used about the importance of the position in the window for predicting the phoneme of the central letter. Without being an expert on linguistics, it is obvious that the central letter in the seven- letter window used is the most important feature in the phoneme prediction. The distance measure proposed by equation 7·1 ignores this fact. Any letter in vector **x** that differs from the corresponding letter in vector **x'** increases the distance measure by the same amount.

Objective functions exist to measure the relative importance of features. These measures are used in feature selection and are called *probabilistic dependence measures*. One class of objective functions is based on the *entropy* function proposed by Shannon [Guiasu 1977, chapter 1,2 and 20].

The concept on which all measures are based is the assumption that pattern **x** and class ω are two random variables. When **x** is observed the *a posteriori* probabilities $P(\omega_i \mid \mathbf{x})$ are computed to determine how much information has been gained from this experiment. If all classes are equally likely then no information is gained and the entropy is maximum. The entropy function can be used to assess this dependency between features and classes. An example of such measure is the *mutual information* defined by:

$$J_s(\mathbf{x}) = \sum_{\mathbf{x}} \sum_{\omega} P(\mathbf{x},\ \omega) \log_2 \frac{P(\mathbf{x},\ \omega)}{P(\mathbf{x})P(\omega)} \qquad (7\cdot2)$$

This function measures the information gain or equivalently the reduction of entropy. The first sum in this equation sums all possible states of **x** and the second sum is concerning all possible classes. A lower bound of this function is zero, if **x** and ω are independent then the fraction is one and the logarithm is always zero. If $\mathbf{x}=\omega_i$ than the information gain is maximal and no uncertainty is left. This upper bound is equal to the entropy in $P(\omega_i)$.

In [Lucassen 1984] this measure is applied to the text to phoneme problem. The feature's $\mathbf{x}_i$ are the letters in the window (figure 7·1) used to classify the central letter ($\mathbf{x}_0$). The feature's $\mathbf{x}_{-1}$, $\mathbf{x}_{-2}$ specify the letters before the central letter and $\mathbf{x}_1$ $\mathbf{x}_2$ the letters after the central letter. In this paper a word corpus of 70,000 words was collected, that contained the 65,000 most commonly used words together with 5,000 regularly inflected forms. The information-gain measured as a function of the position is collected in table 7·3. The second column is the information gain of the *individual* features and the third column is the information gain when features are added as indicated by column order. The upper bound or entropy in $P(\omega_i)$ was reported to be 4.34.

Table 7·3: The mutual information of letters in a word as a function of position in the window used for the phoneme classification.

i	Mutual information $J_S(x_i)$	Cumulative information $J_S(\mathbf{x})$
0	2.993	2.993
1	0.809	3.440
-1	0.636	3.797
2	0.351	4.022
-2	0.217	4.130
3	0.197	4.180
-3	0.130	4.199
4	0.122	4.214
-4	0.110	4.220

This result inspired Sejnowski to use a window of seven letters. The gain in information of using nine letters is small (4.220-4.199=0.021) compared to the total entropy of 4·34. Finally these results are used to propose the following distance measure that is based on the results listed in table 7·3:

$$e_1(\mathbf{x},\mathbf{x}') = \sqrt{\sum_{i=-3}^{3}[J_s(\mathbf{x}_i)\,|\,\mathbf{x}_i - \mathbf{x}'_i\,|]^2} \qquad (7\cdot3)$$

The mutual information is used to weight the different errors and clearly the central letter has become the most important letter now. It is questionable if the values of table 7·3 can be used directly to scale to feature space. This is evaluated in the next subsection. Note that the number of different distances that are distinguished in this space, is increased to 128 levels.

7·2·5 Statistical pattern recognition applied to NETtalk

Two classifiers that also reveal some information about structure in the data are the *nearest-mean* and *k-nearest neighbor* classifiers. For both classifiers a notion of how distances are measured in feature space is important. The nearest mean classifier uses one prototype for every class that is the mean vector of all vectors in the training set belonging to the same class. An unknown sample is classified to the closest prototype.

The family of k-nearest neighbor classifiers regards every sample in the training set as a prototype. For an unknown sample, the k samples of the training set that are nearest in distance from this sample are selected. The sample is classified to the class that is the majority in the set of k samples.

A data set for which the performance of the nearest mean classifier is close to Bayes optimal is a data set that consists of well-defined clusters. A certain compactness and symmetry exist due to the fact the mean value is assumed to be representative to describe each class. This approach fails if samples of the same class are found in more

than one cluster or samples are not clustered at all. In these situations the k-nearest neighbor classifier is expected to perform much better. The optimal value of k indicates the importance of local structure in the data.

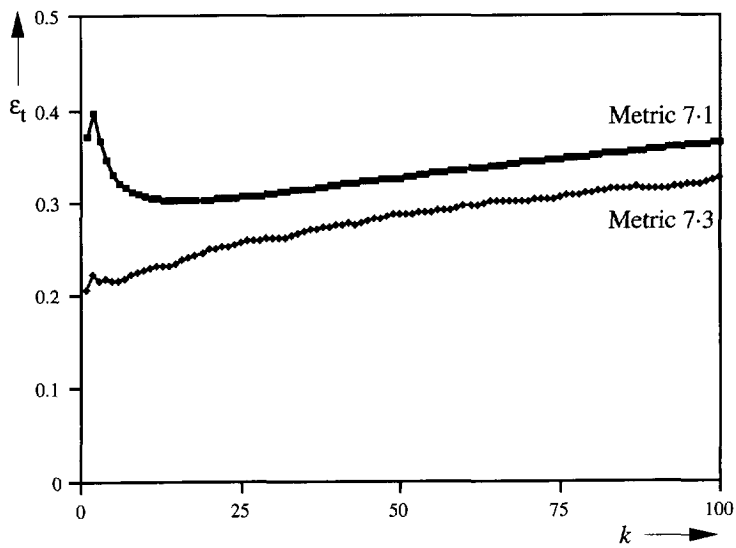

Figure 7·5: The measured performance of the k-nearest neighbor classifier as function of the number of neighbors k. The 1000 common used words are used as training data. The test set is the dictionary word corpus containing 20,000 entries.

The Euclidean mean of every class was calculated using the training set and the classification error of the nearest mean on the test set is listed in table 7·4. The a-priori probability of the silent phoneme (number 52) is with $P(\omega_{52}) \approx 0.148$ the maximum of figure 7·4. A classifier that responds with this class for every input thus reaches a performance of $\varepsilon \approx 0.852$. The result of the nearest mean classifier using metric 7·1 is significantly better and it can be concluded that indeed clustering of the classes is present. The classification error is slightly increased when metric 7·3 is used.

Table 7·4: The classification results of the nearest mean and nearest neighbor classifiers for the two different metrics given by 7·1 and 7·3.

Method	Metric e_H Eq. (7·1) $\hat{\varepsilon}_t$	Metric e_I Eq. (7·3) $\hat{\varepsilon}_t$
Nearest mean	0.388	0.443
1-Nearest neighbor	0.372	0.204
17-Nearest neighbor	0.302	0.239

The performance of the k-nearest neighbor classifier as function of k is plotted in figure 7·5. The optimal number of neighbors for metric 7·1 appeared to be 17 (see table

7·1). For metric 7·3 the nearest neighbor classifier seems to be better ($\hat{\varepsilon}_t \approx 0.204$) and reaches a performance that is almost 4% better than the *best* duplicated network listed in table 7·1 ($\hat{\varepsilon}_t \approx 0.241$). The improvement of the nearest neighbor classifier by using a different metric is an indication that this classifier is suffering from small sample size problems (see [Fukunaga 1984]).

In [Fukunaga 1984] the dependency of the nearest neighbor performance on the measure of nearness (or metric) is addressed. He notes that this dependency is a small sample problem because the nearest neighbor approach is asymptotically Bayes optimal. As sample size increases, the metric becomes less important and the performance convergences to the asymptotic risk. The task is to find a distance-measure so that the finite sample performance is close to the Bayes performance.

The approach followed in [Fukunaga 1984] is to optimize a quadratic distance measure such that the difference between the finite sample performance and the asymptotic performance is minimized in mean square error sense. We did not follow this approach but used the entropy measure to define a linear distance measure. It is not claimed here that this measure is optimal, but this simple improvement of the nearest neighbor classifier outperformed the generalization capabilities of the neural network classifier.

7·2·6 A discussion of the NETtalk application

The investigation of the NETtalk application revealed a number of interesting observations that are important to understand the successes reported in neural networks (see table 6·1). The modification of the training algorithm and the representation of the input surely influences the sample size and generalization discussion. The results of chapter 6, specially figures 6·6 and 6·7, indicate that the classifier's capacity is dependent on the number of training sweeps and a possible correlation that is present in the data. In the NETtalk application a modified training algorithm is used limiting the capacity together with a high correlation in input space.

The main results reported in [Sejnowski 1986, 1987] could be duplicated, although not all experiments were reproduced exactly. A classifier with similar characteristics was obtained but a different number of updates was needed. Furthermore the perfect-match results showed a performance of $\hat{\varepsilon}_t = 1$ which does not agree with the original publication.

Initialization of the weights to random values seemed to be an important source of randomness and confirmed the conclusions of chapter 5. A different *randomly* chosen training set resulted in a network classifier that performed equally to a network trained with the 1000 most commonly used words, if the sensitivity to the initial weights is taken into account. The most important conclusion from a pattern recognition point of view is that the nearest neighbor classifier improved by a simple modification of the distance measure outperformed the neural classifier.

7·3 A comparison of three traditional data sets

7·3·1 Introduction

In this section the performances of various network classifiers and two traditional classifiers on real data sets are investigated. The nearest mean (N-Mean) and nearest neighbor (1-NN) classifiers are chosen because of arguments similar to section 7·2·5. Three data sets that have been used extensively in the past are Fisher's Iris set [Fisher 1936] and two sets extracted from Munson's hand printed character database [Jain 1988b].

The IRIS data set (see appendix A) consists of 150 4-dimensional patterns from three classes. It contains 4 measurements on three species of 50 flowers each. In [Jain 1988b] it is shown that the linear (2·20) and the Parzen window classifiers are the best on this problem with an estimated leave-one-out error of $\hat{\varepsilon} \approx 0.013$.

The IMOX data set contains 192, eight dimensional patterns of the letters I, M, O and X. The features are measurements on these characters derived from Munson's database. The classification results are not as low as for the Iris data set and in this case [Jain 1988b] reports that the 1-NN and quadratic (2·24) classifiers show the best results, with an estimated leave-one-out error of $\hat{\varepsilon} \approx 0.032$.

A second set derived from Munson's character database is the 80X set. Similar to the IMOX data this set contains 45 patterns that are measurements of the letters 8, 0 and X. This set is very sparse and large errors are measured. The 1-NN and linear classifiers are the best and [Jain 1988b] reports a leave-one-out error of $\hat{\varepsilon} \approx 0.066$.

7·3·2 Experiments and results

The limited number of samples in all three data sets, require that an optimal use of the available information must be made to construct and test the classifiers (section 2·4·1). The authors in [Jain 1988b] used the leave-one-out and bootstrap method, that are methods to make an optimal use of the available knowledge. The goal of the experiments here is to compare neural classifiers with the nearest-mean and nearest-neighbor classifier. This comparison is more important than the optimal use of the data so the following procedure is chosen.

A data set was randomly partitioned into two equal sized sets. One set was used to train the classifier and the other set was used to evaluate the performance. This procedure was repeated ten times and the average performance and standard deviation were calculated. The nearest-mean and nearest neighbor classifiers both used the simple Euclidean metric. For the neural classifier, different numbers of hidden units were applied. The networks were optimized using the conjugate gradient descent method (see section 4·3·3) with 100 line minimizations and the initial random weights were chosen uniform between -0.1 and 0.1.

Table 7·5: The measured average performances for the nearest-mean, nearest neighbor and four neural classifiers.

IRIS	Method					
Measure	N-Mean	1-NN	*4-3-3*	*4-7-3*	*4-14-3*	*4-21-3*
Mean ε_t	0.084	0.053	0.428	0.163	0.075	0.039
Standard dev. ε_t	0.025	0.021	0.131	0.153	0.109	0.011
IMOX						
Measure	N-Mean	1-NN	*8-3-4*	*8-7-4*	*8-14-4*	*8-21-4*
Mean ε_t	0.109	0.060	0.268	0.135	0.109	0.115
Standard dev. ε_t	0.033	0.018	0.141	0.040	0.023	0.029
80X						
Measure	N-Mean	1-NN	*8-3-3*	*8-7-3*	*8-14-3*	*8-21-3*
Mean ε_t	0.113	0.091	0.317	0.226	0.209	0.183
Standard dev. ε_t	0.062	0.043	0.152	0.073	0.073	0.073

The simulations were performed on a Sun SparcStation 1+ using the artificial neural network library ANNLIB (chapter 8), together with the pattern recognition extension SPRLIB. The results of all simulations are listed in table 7·5.

7·3·3 Discussion on the classification results of IRIS, IMOX, 80X

The results based on the leave-one-out procedure that were published in [Jain 1988b] (see section 7·3·1) are clearly better than those listed in table 7·5. This is due to the better use of the available limited information. It is questionable if a leave-one-out procedure is possible with feed forward classifiers because of the sensitivity to the initial weights. As stated before, these experiments are meant to compare neural classifiers, especially considering the small sample behavior for real-world applications.

The two traditional methods outperformed the neural classifiers except for the *4-21-3* classifier applied to the Iris data. In most cases the nearest mean classifier performs equal or better than the neural classifiers, together with a equal or smaller observed standard deviation in the results. This is probably caused by a combination of a larger sensitivity to small sample deviations together with the initial weight problem for the neural classifier.

The average error measured for the *4-21-3* classifier and the Iris data set is interesting. The number of training samples is 75 compared to a total of 171 trainable parameters in the network. This example fits well into the list of remarkable applications of table 6·1.

I would like to emphasize that the neural classifiers are applied without a careful tuning of the training method, choice of initial weights, number of line-minimizations and optimal architecture. Sometimes a careful tuning of the setup can improve the network performance. The approach, however, has the risk that the test set becomes the training set because the tuning is often guided by the test set results.

7·4 A comparison of three other data sets

7·4·1 Introduction

In this section three additional data sets are compared using the same approach as in the previous section. The first data set (BLOOD) is published by the American Statistical Association (ASA) and was used at their annual meeting in August 1982 in Cincinnati. Each vendor or provider of statistical graphics software participating in the exposition was encouraged to analyze this data set using their software. The tabular, graphical and text output illustrating the use of graphics in their analysis was summarized to inform the ASA members about the capabilities and uses of computer graphics in statistical work [Cox 1982].

The BLOOD data (see appendix A) arose from a study to develop screening methods to identify carriers of a rare genetic disorder. Four measurements were made on blood samples. Experts in the field have noted that young people tend to have higher measurements, so I used the patient age as fifth measurement. The laboratory that prepared the measurements was worried that there may be a systematic drift over time in their measurement process. I ignored this warning and randomly shuffled the order of the samples which might lead to higher classification errors.

Because the disease is rare there are only a few carriers of the disease from whom data is available. The original data contained 209 observations (134 for *non-carriers* and 75 for *carriers*). I removed the samples with missing data fields which lead to 194 samples containing 127 non-carriers and 67 carriers.

The second data set is a collection of sonar signals (SONAR) bounced off a metal cylinder and of a roughly cylindrical rock [Gorman 1988]. The signals have energies in each of 60 wavebands. The patterns were ordered in time as a boat moved over the objects and hence also by incidence angle. A total of 208 samples was collected of which 111 samples were reflections from the metal cylinder and 97 from the rock.

In the original publication [Gorman 1988] two training sets were used. One training set was a random sampling of the complete set. The best classification result was a neural classifier with 12 hidden units with an estimated performance of $\hat{\varepsilon}_t \approx 0.153$. The second training set was sampled such that the incidence angles were equally sampled in space. As a result of this sampling the remaining samples that were used as test set were *artificially* similar. The reported increase in performance of $\hat{\varepsilon}_t \approx 0.096$ is suspicious.

The experiments of [Gorman 1988] have been repeated by [Ripley 1992]. His results seem to be almost similar to the original paper. He also noticed the similarity of the training set and testing set that were used for the second partitioning of the complete set. The nearest neighbor approach outperforms the neural network using this selection of the training set. For the randomly selected training set the neural network with 12 hidden units performed better than the nearest neighbor approach.

A second set of experiments by [Ripley 1992] indicates that the claimed generalization is doubtful. He re-scaled the channels to unit within-class variance of the training set. The classification results of both neural network and nearest neighbor classifiers deteriorated to $\hat{\varepsilon}_t \approx 0.41$

The last set is a collection of 214 fragments of glass (noted by GLASS) used for a the study that was motivated by criminological investigation (see appendix A). At the scene of the crime, the glass left can be used as evidence if it can be correctly identified. Of each glass specimen the refractive index and the weight percentages of the following oxides are measured: Na, Mg, Al, Si, K, Ca, Ba and Fe. The fragments were originally classed as seven types, but some are very infrequent or absent. I re-grouped them, inspired by [Ripley 1992], in window float glass (70), window non-float glass (76), vehicle window glass (17) and other (51).

The results reported by [Ripley 1992] show large classification errors. The nearest neighbor classifier is best with an error of $\hat{\varepsilon}_t \approx 0.26$. The best neural classifier with 6 hidden units reached a performance of $\hat{\varepsilon}_t \approx 0.39$.

7·4·2 Experiments and results

The experimental setup is similar to section 7·3·2. A data set is randomly partitioned into two equal sized sets. One set is used to train the classifier and the other set is used to evaluate the performance. This procedure is repeated ten times and the average performance and standard deviation are calculated. The nearest-mean and nearest neighbor classifiers both used the simple Euclidean metric. For the neural classifier, different numbers of hidden units were applied. The networks were optimized using the conjugate gradient descent method (see section 4·3·3) with 100 line minimizations and the initial random weights were chosen uniformly between -0.1 and 0.1.

Table 7·6: The measured average performances of the nearest-mean, nearest neighbor and neural classifiers for data sets BLOOD, SONAR and GLASS.

BLOOD			Method			
Measure	N-Mean	1-NN	*5-3-1*	*5-7-1*	*5-14-1*	*5-21-1*
Mean ε_t	0.168	0.156	0.284	0.257	0.227	0.260
Standard dev. ε_t	0.035	0.031	0.040	0.035	0.038	0.033
SONAR			Method			
Measure	N-Mean	1-NN	*60-3-1*	*60-7-1*	*60-14-1*	*60-21-1*
Mean ε_t	0.348	0.204	0.215	0.231	0.208	0.223
Standard dev. ε_t	0.062	0.023	0.044	0.042	0.022	0.034
GLASS			Method			
Measure	N-Mean	1-NN	*9-3-4*	*9-7-4*	*9-14-4*	*9-21-4*
Mean ε_t	0.569	0.293	0.733	0.664	0.652	0.612
Standard dev. ε_t	0.049	0.022	0.126	0.119	0.081	0.106

The results are tabulated in table 7·6 and the simulations were performed on a Sun SparcStation1 using the artificial neural networks library of chapter 8. The nearest-mean and nearest-neighbor classification results were obtained with the pattern recognition library SPRLIB [Schmidt 1993a].

7·4·3 Discussion of the comparison of BLOOD, SONAR, GLASS

The two traditional methods again outperform the feed forward networks, especially the simple nearest neighbor classifier. For the SONAR application the network with 14 hidden units is close in performance to the nearest neighbor classifier. This architecture seems to also be the best choice of the feed forward classifiers evaluated for the BLOOD set but the generalization error is much higher than the nearest mean approach. Note the small difference between the nearest mean and nearest neighbor results for this data.

The GLASS identification problem is a complicated task. The large difference between the nearest mean and nearest neighbor classifier indicates that local information is important. The inspection of this data, that is not reported here, with a multi-dimensional plotting package supports this conclusion. Linear structures are observed too in this data so a nearest neighbor with optimized metric might perform better. The *9-21-4* network classifier does not seem to be complex enough because its performance is hardly better than classifying all samples to window non-float glass (76), which is the class with the largest a-priori probability. This is confirmed by additional experiments that are not tabulated here.

7·5 Discussion

In this chapter seven different applications were investigated where neural network classifiers were compared to the nearest-mean and nearest-neighbor classifiers. The conclusion of this comparison is that except for the Iris data set from [Fisher 1936], the two traditional methods are superior. For the NETtalk problem the nearest-neighbor classifier needed to be enhanced by a simple improvement of the metric, a small sample size problem of this classifier [Fukunaga 1984].

I admit that the feed forward classifiers that are used in sections 7·3 and 7·4 are not carefully optimized to the specific application. A fine-tuning of the architecture, training method and all other possible design issues listed section 5·3 was omitted. Additional experiments, that are not reported here, indicate that the GLASS results of 7·4 are improved if the number of hidden units and line minimizations are increased.

One could argue that the followed procedure is a naive approach to compare the suitability of neural networks to real problems. A fine-tuning of the training procedure by using the test results is a well-known trap in pattern recognition. The test set has

now become the design set as well. The performance measured with this set is a re-substitution estimate. This estimate is biased and therefore leads easily to optimistic conclusions about this new type of classifier.

One should be aware of this trap when reading publications of neural network applications. In most articles, [Sejnowski 1986] for example, the authors are vague about how they found the training parameters and architecture. The classification of chromosomes using neural networks by [Errington 1993] is a good example of researchers that motivate how they found their network architecture and training parameters.

On the other-hand the approach that was followed by me probably leads to pessimistic conclusions. The network architecture and the optimization method influence the classifier's capacity (chapter 6). This classifier's complexity should match the sample size and data complexity. With the practical application of neural networks, the importance of the issues discussed in chapter 5 becomes apparent.

8

A Software Library for Neural Networks

8·1 Introduction

An obvious starting point for neural network research is the purchase of a commercially available neural network simulator, see for example [Korn 1989], [NeuralWorks 1991] or one of the 66 other simulators in the overview by Murre [Murre 1992]. Generally, the simulator is equipped with an excellent graphical user interface and many network paradigms are supported. Building an application, using one of the standard built-in methods is easy and the first results can be obtained after hours of working rather than days.

Problems arise when the application does not seem to work with the standard solutions and when it is not understood why. Generally, the possibility to adapt the algorithm or to add a newly developed algorithm to the simulator is either not available or very difficult to perform. Even if facilities are available, the documentation is complicated and the user has to use strictly prescribed rules to access or add new parameters. More time is then spent in meeting the requirements of the simulator, than in doing the creative work of algorithm design or building the application. Apparently, this seems to be a problem of many available neural network simulators.

In the environment where most of my work was done, that of pattern recognition research, this appeared to be a crucial problem in performing any research. In most cases the goal is not to solve a particular pattern recognition task, but to gain more understanding on the advantages and limitations of neural network classifiers. The influence of various learning rules and strategies, network topologies, network paradigms and various kinds of training data on the classification performance are being investigated. For my purpose, the simulation environment should therefore offer much more flexibility in accessing low-level data than is generally possible in commercially available simulation environments.

On the other hand, the obvious alternative of implementing an algorithm with custom made software does not seem to be very attractive either, since a large amount of tedious and laborious coding is involved. Also, it is not sure that code that was developed today, can be reused for new implementations and/or algorithms tomorrow. This is rather inefficient, since a large part of a simulation program generally consists of code for disk I/O, presentation to the user, and estimation of performance.

An intermediate approach that is advocated in this chapter considers a library of subroutines that operate on a set of powerful data structures as the ideal vehicle to develop and simulate neural network algorithms. Among these subroutines should be general ones for performing disk I/O, error handling as well as a set of standard learning algorithms. The advantage of this solution is that the same functionality is offered as with a purchased simulator, but with much more flexibility. The price that is paid, however, is that the user interface is not as refined. This is something that is easily overcome in a research environment.

The subject of this chapter is the design philosophy and description of a simulation environment that was used to perform all experiments that were reported in this thesis. The simulation library is called *ANNLIB* [Schmidt 1993a] and it is the result of a cooperation between M.A. Kraaijveld and me.

This library became part of a similar statistical pattern recognition library, called *SPRLIB*, that was developed by me to allow a comparison and integration of traditional pattern recognition and neural networks. The integration of both libraries imposed a few requirements on ANNLIB. The requirements and some implementation aspects will be discussed in section 8·2.

The statistical pattern recognition library (SPRLIB) will briefly be discussed in section 8·3. This is only a quick summary of this library and only those parts that are important for ANNLIB will be discussed in detail.

The essential part of our neural network simulation environment is the flexible network data structure that is powerful enough to hold all kinds of network paradigms in an efficient way and allows the user to control and to manipulate all parameters. In section 8·4 the requirements of this data structure together with the implementation details will be discussed.

A network data structure alone is useless so in section 8·5 an explanation of all the functions will be given such as the currently supported network algorithms. In section 8·6 a number of tools are discussed. Some tools are designed to interface the library to other software. A generic set-up and a number of examples that facilitate the user to build his/her own application are also discussed, together with an example source listing. This chapter ends with a discussion concerning the limitations, availability and use of the described simulation library.

8·2 Requirements and implementation aspects

8·2·1 System requirements

Before the decision to build a simulation library was made, it was realized that the context of performing research on algorithms dictates a number of specific system requirements. The most important are stated in the following:

- Because being part of a pattern recognition group, the interest in neural networks is mainly focused on its use as a pattern classifier. One of the basic questions in this context is how a neural network compares to a traditional statistical pattern classifier. The first requirement of the system is that is must easily allow a comparison of both approaches.
- A comparison of network classifiers with traditional methods could lead to the conclusion that for certain tasks neural network classifiers are better, or that the use of hybrid systems can improve a classification problem. This leads to the requirement that it must be possible to create hybrid systems.
- Solving a certain application is only a small part of our interest. I would like to have a system that offers the maximum flexibility to implement new ideas, new algorithms or improvements to established techniques. This implies that the system should be an open system, so that the user can have access to all parameters and can alter parts of code or add new code. This last requirement implies that the system maintenance should be supported, preferably by special tools.
- Although it is not the main purpose, building standalone applications must be possible with this system. Another requirement is the possibility to integrate the library with other software, for example image processing packages.
- It is important that an environment is portable to different machines and architectures without much reprogramming. However, neural network experiments need large computational resources, especially speed and memory, so it is reasonable to restrict the target machines to the more powerful workstation computers or high-end personal computers.
- An environment with substantial flexibility and possibilities will easily result in a system that is difficult to start with. A novice user will have difficulties building his or her first application. Therefore a number of starting points or examples should be available that can be helpful in getting started with the system.

8·2·2 ANNLIB implementation

As a result of the previous list of how a solution should look, our choice was to build a library of neural network functions. The final simulation environment consists of a compiled C-library with about 70 neural network functions and 100 numerical and statistical pattern recognition functions. The portability is ensured by writing the package in traditional Kernighan and Ritchie C code, see [Kernighan 1978]. UNIX was chosen as the platform to develop the library for its good support of software development.

A standardized set of powerful data structures offers the flexibility that is required for various types of networks, together with easy access to its parameters. A large set of routines operates on these data structures. The user has to write his or her own main program, which calls the library functions.

One of the drawbacks of this approach is that a graphical user interface is lost, that is available in other environments. This is partially solved by writing interfaces to packages for data analysis and graphics facilities. Typical interfaces are available for Mathematica [Wolfram 1992], plot and graph for UNIX systems, spreadsheets and business graphics software.

8·3 The Statistical Pattern Recognition library

A set-up similar to the neural network simulation environment had already been developed for statistical pattern recognition experiments [Schmidt 1993a]. Because this library (SPRLIB) was developed before an interest in neural networks emerged, some of the important data structures used in the neural network library are the same as those developed for SPRLIB.

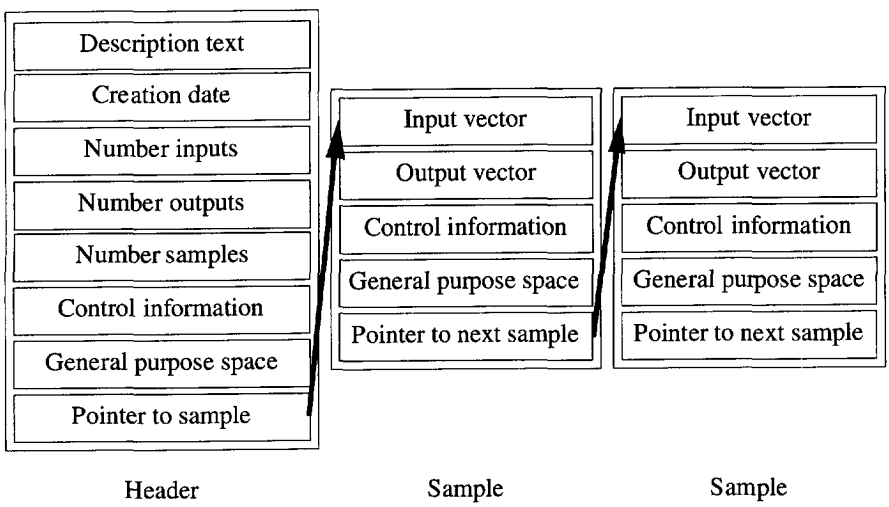

Figure 8·1: The data set data structure.

In this section the data structures and supporting library routines will be explained that are necessary for the use of functions from the neural network library. Furthermore a list with the implemented statistical pattern recognition paradigms is found at the end of this section.

8·3·1 Important data structures in SPRLIB

The first important data structure is imposed by the decision to use the numerical routines from [Press 1982], *Numerical Recipes in C*, for all numerical calculations. These routines use a special data structure for matrices and vectors. For a detailed explanation, see chapter 1 of [Press 1982].

The second important data structure adopted from SPRLIB is the way that data sets are stored. A data set consists of a data set header with a linked list of all sample. The header contains general information about the data set such as the creation date, a description string, dimensions of input and output vectors (the targets **t** in neural networks) and number of samples. Space is reserved, a pointer for example, for the user to hold additional data for his or her own purpose. See figure 8·1.

Attached to the header is a linked list of data points that holds the actual data, pairs of input and corresponding targets, together with control information. As in the data set header, space is reserved for the user to store his own information, if needed.

8·3·2 Important support functions in SPRLIB

Some additional functions to the *Numerical Recipes in C* library are present such as matrix multiplication, matrix-vector multiplication, vector-vector multiplication, matrix-transpose, matrix-copy and matrix-inverse. Furthermore, two functions are implemented to print out matrices and vectors in a formatted way.

The library functions to support the user with data set handling can be roughly divided into three groups. The first group of functions is for creating and deleting data sets and the optimization of the memory management associated with handling data sets.

The second group of functions is for storing and retrieval of data sets. Data sets are stored on disk in a straightforward ASCII format. If data sets are very large, the library supports adaptive Lempel-Ziv compression of these files for UNIX machines, transparent to the user.

The last group of functions allows the user to scale the inputs and targets in any arbitrary way. Some methods depend on the evaluation order of the samples. A function to randomize the order in which the samples are stored is available too.

8·3·3 Pattern recognition algorithms in SPRLIB

Here follows a brief list of some of the pattern recognition methods implemented in the pattern recognition part of the library:

- Mean and covariance estimation.
- Within and between scatters estimation.
- Principal component analysis.
- Random number generator for arbitrary multidimensional normal distributions.
- Fisher linear discriminant function.
- K-nearest neighbor classifier.
- Nearest mean classifier.
- Basic isodata clustering method.
- Parzen window probability estimation and classification.

8·4 Data structures in ANNLIB

A key issue in the design of a neural network simulation library is the design of a proper set of data structures. Powerful data structures should allow the incorporation and implementation of a wide variety of neural network paradigms, whereas the clarity and consistency of the data structures are an essential condition to overcome delay for novice users. In this section a number of such design considerations of neural network data structures are described, followed by an explanation of the implementation in ANNLIB.

8·4·1 Considerations on network representation

The design of a set of data structures for neural network simulation has shown that a large number of aspects appear to be crucial for the success of the environment. Among these are elements such as the consistency of the data structures, implementation aspects like the memory usage and the speed of algorithms, the robustness, etc. A non-complete list of such aspects is the following.

- Flexibility: A difficult issue in the design is that there is a need to anticipate network paradigms that do not even exist yet. It should certainly be avoided that a slight modification of an existing algorithm or paradigm imposes a change in a key data structure, since that will imply a major programming effort. On the other hand, too much flexibility may lead to over-complex data structures that are difficult to understand. The key data structures should therefore be so powerful that all standard network algorithms and paradigms can be implemented, which means the requirements of a large number of paradigms should be taken into account. Additional complexity and/or flexibility can then be added by the incorporation of pointers that the user can use to point at any user defined structure. Also pointers to functions within the data structure should allow the use of algorithms that are specific for a certain context. A change of context, then only requires the change of these pointers, instead of the complete data structures.
- Clarity, consistency and intuitiveness: Clear, consistent and intuitive data structures make a large part of the available algorithms self-explanatory. This largely helps the user in obtaining the understanding that is required to work with the library.
- Implementation issues: A well-known fact is that good data structures can prevent heavy computations and may greatly influence the memory requirements of the software. The potential non-regular topology of a network here advocates the choice for linked lists of structures instead of arrays. This also has the advantage that large amounts of address computations are avoided.
- Robustness: A certain amount of redundancy in the data helps the user and the software to detect inconsistencies in the data. Therefore, some variables that are

implicit in the data structures (for example the number of units in a network), should also be stored explicitly, to increase the robustness of the software.

• User convenience: Unique identification numbers and the availability of fields in which the user can store verbal descriptions of the role or functionality of the data structures help the user to interact with the software. Also, standard ASCII file formats largely increases the accessibility of the data in file format. The disadvantage of large requirements regarding disk space can then be solved with standard compression facilities.

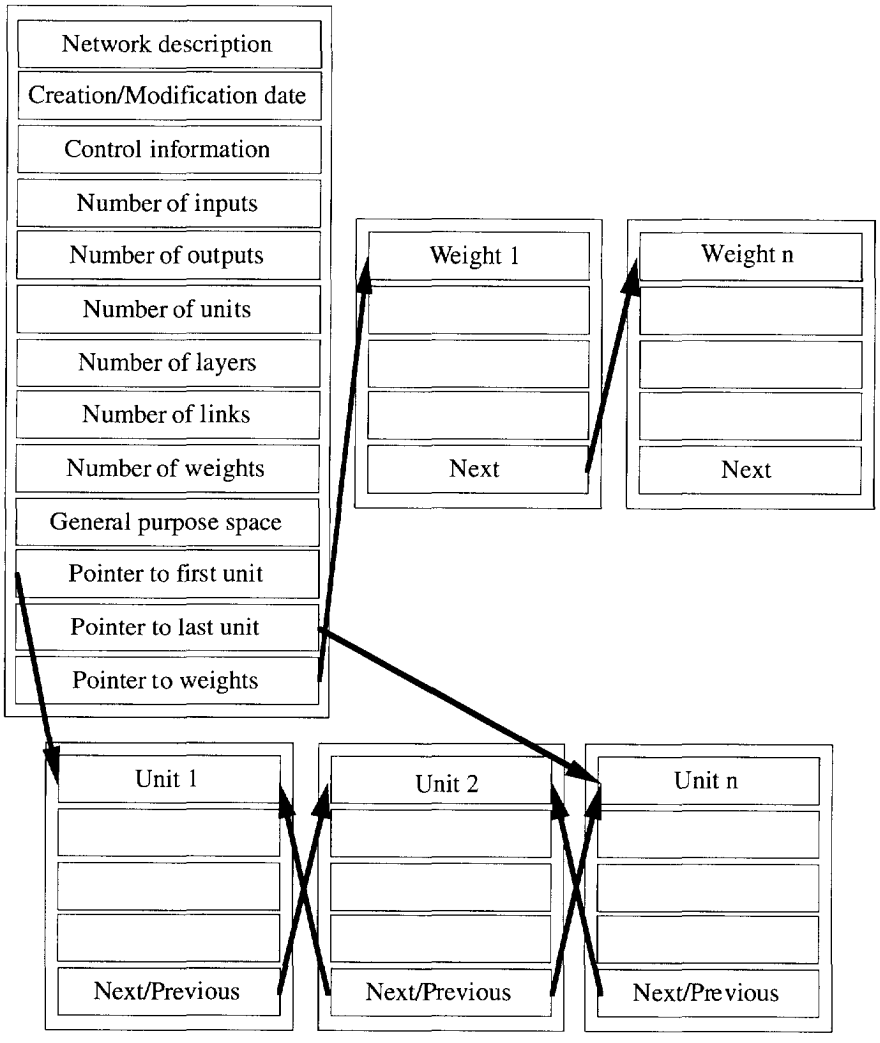

Figure 8·2: The top network data structure.

8·4·2 Implementation in ANNLIB

In this section the data structures that represent a network in ANNLIB will be explained in detail. A network is represented by five different structures: the network, the unit, the link, the weight, and the unit value. These structures are described in this section.

The top level network data structure

The top of the network data structure is a network header. All network related functions use a pointer to this data structure on input and all variables can be accessed from this structure. The following information can be found.

A network description text, and the creation and modification date are there for the user's convenience. The control information is important, it specifies the type of network and if it is one of the standard network types found in the library or a user defined type of network. Most routines test this field to check if a certain routine may be called with this type of network.

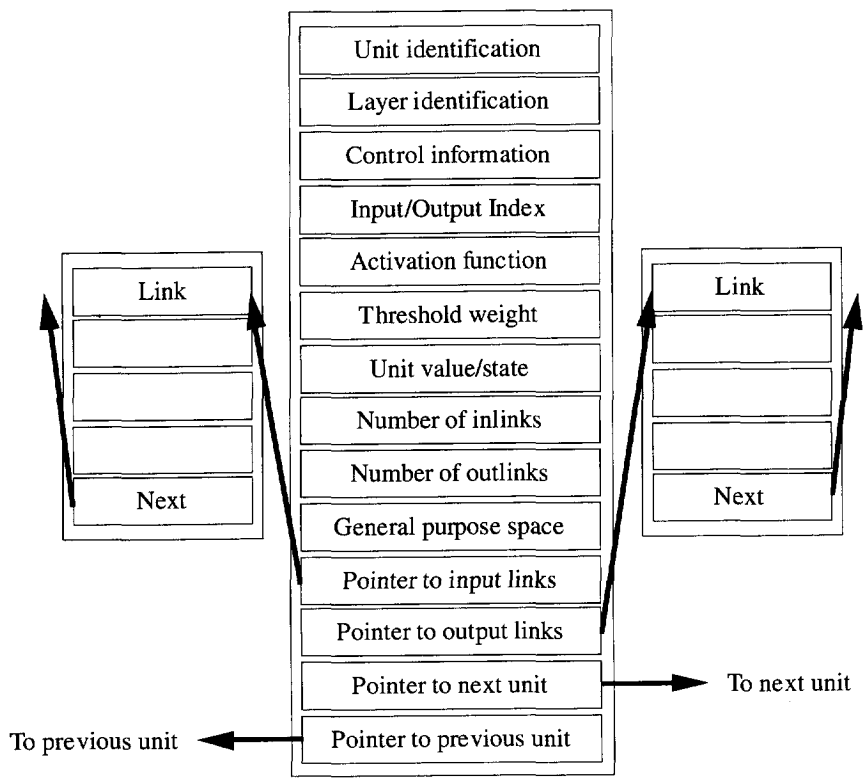

Figure 8·3: The unit data structure.

Furthermore, some general data about the network is stored in this header, like the number of inputs or outputs, the number of units, layers, weights and links in the

network structure. These parameters are only valid if they make sense for the type of network and may be undefined if not appropriate. A general purpose data field allows the user to attach his own data to the network structure. This may either be a long integer, a double floating point or a pointer. The latter allows the user to attach any data to the network data structure.

A pointer to the first and last unit of the double linked list of units gives access to the whole network structure, which will be explained in the next section. A pointer to a linked list of all weights is provided for quick access to the parameters. See figure 8·2 for a schematic diagram of this set-up.

The unit structure

The units together with the links determine the network topology. Connected to each unit is a linked list of links that specifies the units or external input terminals that are the inputs to the unit. A second linked list of links determines the units to which the unit output is transferred. Thus, given a unit in a network all inputs and outputs can immediately be found.

Besides pointers to the links, the unit contains a unique identification number (which can be used for searching), a layer identification number (if appropriate) and control information. This control information specifies the type of unit (for example input, hidden or output) and allows the possibility of disabling a complete unit. The input/output index specifies whether a unit is an input or output unit and which input or output it represents.

A special weight, called the threshold weight is stored in the unit. Its structure is the same as all other weights and is used in some network paradigms. A pointer to the unit activation function and unit transfer function allows different types of units to be mixed in the same network. The standard supported activation functions are the weighted sum of inputs and weights and the Euclidean or squared Euclidean distance of inputs and weights. The user can easily define his own activation function and put the pointer in the unit to point to the new function.

The transfer functions are implemented in a similar way. Standard transfer functions are linear, sigmoidal, tanh and Gaussian. The user can point to a user defined transfer function if needed. For learning methods that require the derivative of the transfer function, a pointer to such a derivative can be stored in the unit structure too.

A counter of the number of input links and the number of output links, although redundant, is added for consistency checks. General purpose data space enables the user to add his own information to this structure. See figure 8·3 for a schematic diagram of this data structure.

The link data structure

The idea of links also appears in the Rochester Connectionist Simulator [Goddard 1988]. The link data structure represents a connection from one processing unit to another unit in a network. Associated with a unit can be input links; which means links associated with the connections coming to the units, and output links: those links that

transfer the unit output to other units. For some network paradigms, therefore, two units are connected by two links representing the same connection and associated with the same weight.

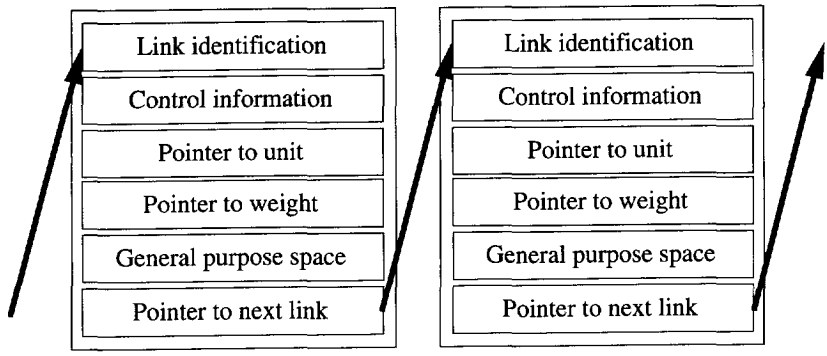

Figure 8-4: The link data structure.

Links are stored in a single linked list attached to the unit. It contains a unique identification number, which can be used for referencing a specific link. Control information specifies the type of link, for example input-link or output-link, and the pointer to the unit to specify to which unit the connection points to. The weight pointer (see next section) points to the structure holding the connection strength.

Again data space is reserved for the user to add his own information to the data and finally a pointer is found to the next link in the list. See figure 8-4.

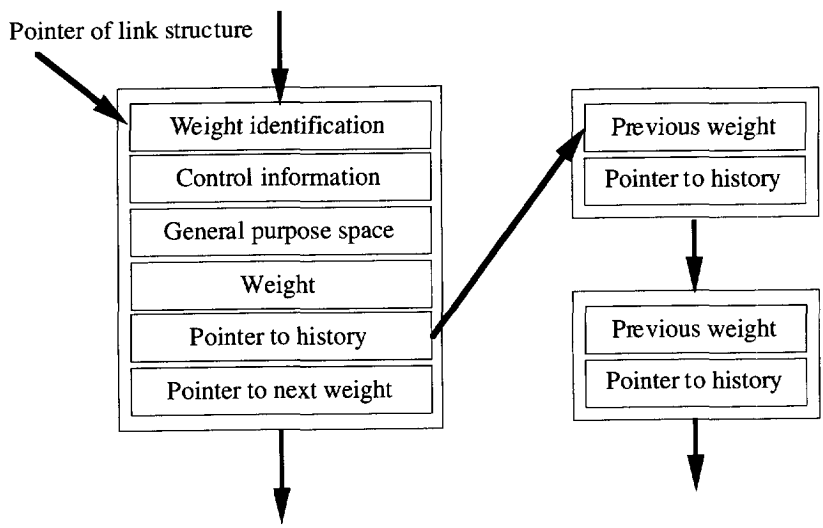

Figure 8-5: The weight structure.

The weight structure

As mentioned before, the weights are arranged in a single linked list of weight structures (figure 8·5). This structure contains a unique weight number and control information. Weights can either be fixed, which means they can not be trained, or variable, which means that they may be updated. The actual weight is stored as a double precision floating point value.

A unique feature of our environment is that the weight structure also holds a pointer to a list of previous weight values. This makes it possible to store the history of the network to study the evolution of the weights during training. General purpose data space allows the user to add her own information to the weight structure.

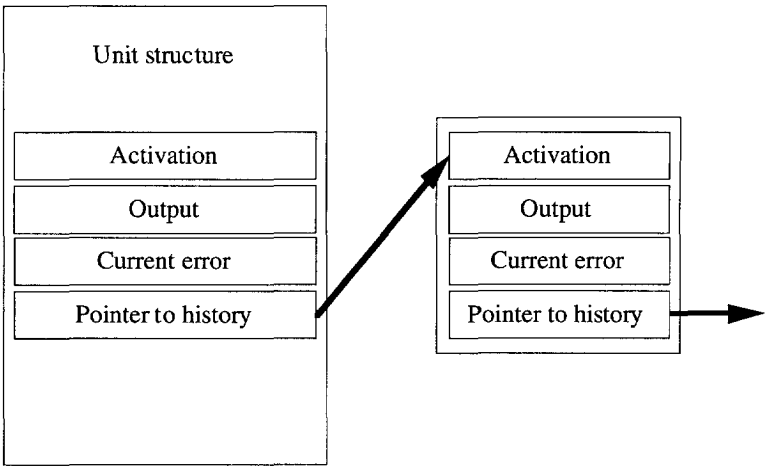

Figure 8·6: The unit value and history values.

The unit value structure

The unit value or state which is a part of the unit structure holds the current activation (see figure 8·6). The activation is a function of the unit inputs, defined by the linked list of input links and the specific activation function. Applied to this activation is the transfer function that determines the unit output. A field called current error is used for some network learning strategies, like back-propagation.

The current activation is found as part of the unit structure. However, a pointer to a history is found here too. This allows the possibility of recording a history of activations.

8·5 Functions and Algorithms in ANNLIB

As with the data structures, the set of available algorithms requires considerable attention in the design. The set of algorithms should consist of a large set of supporting functions together with a set of routines that implement the standard learning

algorithms. In this section, a number of such design considerations are discussed, together with a description of the implementation in ANNLIB.

8·5·1 Considerations on functions and algorithms

A number of the considerations that were mentioned for the data structures, also hold for the set of algorithms and functions to be implemented. Here also the flexibility, the consistency, the clarity and the robustness are important design criteria for the set of algorithms. More important, however, is that all low-level functions that are trivially required, but tedious to develop, are available in the standard set of routines. Disk I/O, allocation of various structures, and the presentation of network data in a properly formatted way to the user are important candidates in this context. Also, to help the user to apply standard algorithms to a problem, the standard set of neural network algorithms should be a part of the library. This has as an important secondary advantage: that the sources of these algorithms can serve as a starting point for the implementation of newly developed algorithms.

Finally, the library should offer a set of routines that exploit all the redundancy in the data structures to verify their consistency.

8·5·2 Supporting network functions in ANNLIB

In this section the library functions that are independent of the network paradigm will be discussed. These functions can roughly be divided in low level and high level routines. The low level routines are intended to support the user who wants to implement his or her own algorithms. The high level routines deal with things that are independent of the type of network.

Low level supporting functions

This class of functions forms only a small set. An important issue is the optimization of memory management. A large amount of data is used, but allocated only as small chunks. Without special memory management this can become a significant limitation for the program speed. The library helps the user to circumvent this problem.

The evaluation of a single unit is a straightforward procedure. Most of the processing is done by the activation function and transfer function. Standard activation and transfer functions are available (see previous section). A low level function for evaluating a unit is supported.

The data structure of the network may be difficult for the novice programmer to navigate. A list of functions to search the data structure for specific units, weights or links help the user around the data structure.

High level supporting functions

The most important group of high level functions is to read and write a network to disk file. A network is written to a disk file using an ASCII format, creating a machine

independent representation. The network structural organization, together with all the weights, are stored in one file.

If network files are large or not loaded very often, the library enables adaptive Lempel-Ziv compression of these files on UNIX machines. A general network loading function makes this transparent to the user. The loading time will be longer due to the run time decompression needed.

A function to check the network consistency, to determine if a loaded network is valid, runs various tests checking the topology of the network. Some heuristic rules are applied to warn about parameters with unrealistic values, for example weights with very large or very small values.

Many network paradigms require random initialization or random perturbation of the weights. Two functions are available for that purpose. The history mechanism explained in section 8·4·2 allows the user to record the evolution of different parameters. A number of functions are available to control this. This option should be used with care because it can lead to very large databases, especially if stored to disk.

The last set of functions is to monitor all network parameters. These are all ASCII based formatted printings of all structures found in the network memory structure.

8·5·3 Training algorithms in ANNLIB

Feed forward networks

The library offers a number of powerful routines for feed forward networks (see [Rumelhart 1986]).

- Network creation and manipulation: A number of routines are available to create and manipulate a network. A feed forward network can easily be created with a library function, allowing any number of layers and any number of hidden units per layer. A routine is also available to add hidden units to an existing network.
- Network evaluation and performance estimation: The evaluation of a feed forward network, for a given input is evaluated with one function call. The performance of a network on a test set can be estimated per sample and for complete data sets. The following measurements are supported: the mean squared error, the maximum error, the minimum error, the gradient, the percentage of the samples correctly classified and a Boolean that is true if all samples are correct.
- Network training: The standard back propagation (see [Rumelhart 1986]) algorithm is implemented with low level functions. This gives the possibility to make changes, such as the update method used in NETtalk (section 7·2·1), without rewriting the whole algorithm. The user who just wants to use the vanilla back propagation, can use the high level routines. This set-up allows a maximum flexibility for experiments.

  Besides the standard back propagation different learning methods can be used to train the network. A method is implemented which uses the *conjugate gradient descent* method (see [Press 1988]) for learning the network weights. The

Levenberg-Marquardt and *BFGS* methods are two well-known methods in numerical optimization (see [Van den Bos 1982]) and can be used for network training too.
For fixed weights in the hidden layers, the network output unit, if chosen linear, can be determined by using a pseudo inverse matrix method. This is a common method in regression analysis and is called the Wiener weight solution in [Widrow 1985]. This pseudo matrix inverse method is supported in the library.
• Network code generation: A network that is trained and is found to be satisfactory can be converted to a Kernighan and Ritchie (see [Kerninghan 1978]) compatible source code equivalent. No loading of a network is needed for the use of a network solution within an application. The library does not need to be linked, a compact and fast solution is generated.

Kohonen networks
The second neural network paradigm that is implemented in the library is the Kohonen Topology Preserving Mapping Algorithm [Kohonen 1982], [Kohonen 1988a], [Kohonen 1988b], [Kohonen 1990]. The following sets of routines are implemented:

• Network creation and manipulation: Kohonen networks with intrinsic dimension ranging from one to four can be generated and manipulated.
• Network training: A Kohonen network can be trained with the standard Kohonen learning algorithm and some of its variants. Among these are the algorithm that only updates the units in a small and limited neighborhood around the unit that is closest to the data input vector and the algorithm that updates all units, but weighed with a function of the distance to the closest unit, see [Kohonen 1988b].

8·6 Supporting software tools

In this section a number of tools are described that are written on top of the library, or facilitate the interface of the library to other packages and environments.

8·6·1 Interfacing to other software packages

The following list of supporting software tools is implemented to import and export data and results to various other software packages:

• Data format conversion: It is important that the user can use the library with data acquired from other sources. To transform data sets to SPRLIB readable format, a program to import data can be used. The user must make minor changes to the source code which determines the format of the data to be transformed. The data format of SPRLIB data sets is straight forward and transformation from SPRLIB format to a generic form is easily done by UNIX commands.

- Interfacing to graphics software: The graphical display of data that is obtained from the training sets or simulation runs of the library is an important way to obtain good insight into the nature of the data. Since there are numerous excellent graphics packages, we chose to interface to these packages rather than writing the graphics ourselves. A tool is available to convert network and training data to formats that can be processed by spreadsheets or business graphics packages and to Mathematica [Wolfram 1992]. Another output format is an *image* that is determined by changing two network inputs and computing the corresponding output. Such an image can then be processed by an image processing package.
- Interfacing to statistical software: A similar interface as was mentioned above was developed to convert data to formats of statistical software packages. This facilitates a convenient way to obtain various statistical measures of the data.

8·6·2 Back propagation program

The most widely used artificial neural network algorithm is surely the back propagation method proposed by [Rumelhart 1986]. A standard implementation is written, together with a large number of special options. This implementation allows the user to experiment with different activation functions, network sizes, collect statistics on learning performances, test different stopping criteria, different initialization conditions, etc. For most applications this support tool will be sufficient and because all source code is available it is easy to make changes.

8·6·3 Generic application set-up

Building an application for some experiments may take some time. Besides understanding the library and how to call all the functions needed, programming has to be done to implement the specific experiments one wants to perform. To help the user to get started, a generic application set-up is written, which can serve as a starting point to build an ANNLIB application. Some trivial things like command line handling and loading and saving of data sets are already taken care of. The user only has to program his experiments and print out the desired statistics.

8·6·4 The example set-ups

The library is fully documented with on-line manual pages. However, if the usage is not completely clear, an example usage can be found for most library functions. These example applications are part of the library and can also serve very well as an initial set-up for a new program. The following example programs are implemented:

- Simple back propagation.
- Two Kohonen examples
- Conjugate gradient descent method

- Levenberg Marquardt method
- Simple perceptron

8·7 An example application

In figure 8·7 an example source listing is presented, here the standard back propagation method is applied to the well-known exor problem. The following steps can be found. First the library is initialized and the learning set is loaded into memory. In the following steps a feed forward network is created with 2 inputs, 4 units in the hidden layer and one output. The initial weights are set to random values between -0.01 and 0.01.

In the next part the actual learning is done. First the back propagation needs to be initialized before training. In the innermost loop the network is trained for 10 cycles and then the network's performance is evaluated. If all training patterns are classified correctly, or if the number of training sweeps exceeds the 1,000,000 the training is stopped. The memory allocated by the initialization of the back propagation is then freed and the network is saved to disk. Finally all other data is cleared and the program exits execution.

8·8 Discussion

In this chapter a neural network simulation environment is presented that is developed for neural network research in a pattern recognition environment. Research in the neural network field often requires the maximum flexibility and efficiency, speed and memory that is offered by high level compiler programming languages only. Most simulation software tools are not designed as an open system and are of limited use for pure research.

The system that has been developed is a library of C functions for artificial neural network simulations. The main feature of this simulation environment is a set of powerful data structures with a large set of supporting functions. There is no special user interface designed; the UNIX programming environment has proved to be a good development environment.

Except for some minor details, like Lempel-Ziv compression of data and network files, the library should compile on any machine with a standard Kernighan and Ritchie [Kernighan 1978] C compiler. At this moment the library runs on Sun 3/4, Stardent 3000, Silicon Graphics Indigo and IBM or compatible personal computer systems.

The choice to build a library and not a standalone simulation tool, easily fulfills all the requirements as listed in section 8·2·1. The C source code provides the full flexibility and efficiency. Furthermore the library can easily be linked or interfaced with other software environments, like image processing packages.

```
#include <stdio.h>
#include <math.h>
#include <sprlib.h>

main(argc, argv)
int argc;
char *argv[];
{
   NET *NetPtr;
   FILE *stream;
   DATASET *LearningSet;
   ERROR_STRUCT ErrorStruct;
   int *NetVect;

   sprinit(INIT_EXIT);  /*Initialize the library*/

   LearningSet = load_dataset("exor.dat");   /*Load learning set*/

   NetVect = ivector(1, 3);        /*Make a 2-4-1 feed forward network*/
   NetVect[1] = 2; NetVect[2] = 4; NetVect[3] = 1;
   NetPtr = create_ff_net(1L, 3, NetVect);
   free_ivector(NetVect, 1, 3);

   unif_rand_net(1, -.01, 0.01, NetPtr);/*Randomize the weights of the net*/

   /*Train the network with backpropagation with default options*/
   {
      long BpOptions = (NO_HISTU | NO_HISTT | NO_HISTW | BPACCUM | BPUPDATE);
      long done = FALSE;
      long TotalCycles = 0L;

      bp_init(NetPtr);             /*Initialize back propagation*/
      while (done == FALSE) {

         /*Perform 10 learning cycles, learning rate = 1, momentum = 0.9*/

         bp_learn(NetPtr, LearningSet, 1.0, 0.9, 10L, BpOptions);

         TotalCycles += 10;

         /*Compute and print the performance of the network*/

         net_perf(NetPtr, LearningSet, &ErrorStruct, 0.1, SIMPLE_PERF);
         printf("Cycle %4d: ", TotalCycles);
         fprintf_error_struct(stdout, &ErrorStruct);

         /*Stop criteria*/
         if ((ErrorStruct.AllCorrect == TRUE) || (TotalCycles > 1000000))
                  done = TRUE;
      }
      bp_free(NetPtr);
   }
   /*Save the network in a file*/

   if ((stream = fopen("exor.net", "w")) == NULL)
      { printf("Can not open file 'exor.net'.\n"); exit(1); }
   fprintf_network(stream, NetPtr, (HISTU | HISTT | HISTW));
   fclose(stream);

   delete_net(NetPtr);             /*Free all structures and quit*/
   delete_dataset(LearningSet);

   sprexit(EXIT_SILENT);      /*Reset library and free internal memory*/

   return 0;
}
```

Figure 8·7: An example source listing

An important requirement of the simulation environment is the possibility to compare and combine neural network techniques with traditional pattern recognition methods. Because the neural network library is part of a statistical pattern recognition library (SPRLIB), this requirement is met.

A less developed part of this system is a graphical interface. A number of tools were developed to interface our system with other packages that allow graphics to obtain insight in the data produced by neural network experiments. Mathematica, for example, is one of these packages.

Until now, approximately 20 people have used our library intensively in practice at three different institutions. After 3 years of practice and being used for a number of publications and dissertations, the system has proved to be a valuable tool for research in the field of neural networks and pattern recognition.

9
Conclusions and Discussion

9·1 Introduction

In this thesis the developments in neuro-engineering were analyzed from a pattern recognition point of view. The multi-layer perceptron is (in my opinion) one of the most important developments in this field and has therefore been chosen as the subject of research. The multi-layer perceptron fits well into the general model of a *statistical* pattern classifier and a large number of applications are reported using the multi-layer perceptron for pattern recognition purpose.

The field of pattern recognition has traditionally been dominated by a mathematical and statistical approach and the classical techniques are therefore inspired by statistical or mathematical arguments. In neuro-engineering the human brain and its neurons are the source of inspiration and this field of research leads to different models. The arguments that lead to the multi-layer perceptron were explained in this thesis and are clearly not based on any assumption about the underlying probability distributions.

Methods that use no prior information about probability distributions are called *non-parametric* techniques in statistical pattern recognition and have received increased attention in recent years. The non-parametric techniques are attractive in practice because in most applications no knowledge is available about the underlying densities. Feed forward classifiers are in that sense interesting for real-world applications.

In *chapter 2* statistical pattern recognition was reviewed and the properties and phenomena that are important in this approach were discussed. The issues discussed in that chapter served as inspiration for the research that was undertaken in this thesis. Here follows a list of properties that have been topic of discussion in the analysis of neural inspired classification systems.

- The *capabilities* or *complexity* of a classifier.
- The problem of estimating the unknown parameters in the classifier.
- The *asymptotic* (infinite) sample size properties.
- The *finite* sample size properties and the expected probability of error.
- The relation between classifiers complexity and the expected probability of error.
- The selection of the best classifier from a number of alternative classifiers.

These issues were discussed for the simple perceptrons in *chapter 3* and well-established theory exists for this type of classifier. The simple perceptrons are only capable of implementing linear boundaries (except for the functional machine) and different training methods exist for training them (see table 3·3).

The asymptotic relation to the Bayes classifier is dependent on the type of perceptron and is summarized in table 3·3. The relation between the classifier's complexity and the probability of error was investigated in section 3·6·1 [Cover 1965] and 6·4·1 [Foley 1972]. The finite sample size properties were discussed in 3·6·2 by means of a Taylor series expansion and a table of measured probabilities of error for different sample sizes, different dimensionality and different Mahalanobis distances.

9·2 Conclusions

In chapters 4, 5 and 6 the properties listed in section 9·1 were investigated for the multi-layer perceptron. The multi-layer perceptrons discussed in *chapter 4* are not limited to linear boundaries and mathematical theory indicates that any continuous function can be approximated by these systems. A less rigorous proof follows by applying the results from [Cover 1965] and confirms the increased capabilities of the multi-layer perceptron to the simple perceptrons.

Training feed-forward classifiers is less obvious because these systems contain large numbers of parameters. A comparison of six gradient based algorithms shows that the back-propagation method and the conjugate gradient descent method are the most appropriate with respect to memory and computational efficiency. The conjugate gradient descent method requires less user interaction and is in that respect much easier to use. Finding a set of stable training parameters for the back propagation method is not always easy. A non-iterative and fast converging algorithm is proposed as alternative. This method has the same limitations as hierarchical classifiers and is expected to be valuable for those situations where hierarchical classifiers are useful.

The asymptotic (infinite sample size) properties easily follow from an analysis similar to the asymptotic properties of the simple perceptron. This analysis shows that the outputs approximate a function of the *a posteriori* probabilities if the network is chosen sufficiently complex. The asymptotic properties show the relationship between the Bayes theory and the network outputs. An interpretation of the function of the hidden units indicates that a scatter related function is optimized and the hidden units can therefore be classified as feature extractors.

The problem of selecting the best classifier from a set of classifiers has been discussed in *chapter 5* and it was shown here that this is an important issue when neural classifiers are applied in practice. A large number of design considerations were discussed and finding a good choice for all these design issues is closely related to the classifier selection problem in pattern recognition. A sensitivity for the initial weights was shown and it is new in pattern recognition that the optimization of the classifier is a

source of randomness. The effect of the number of training sweeps, the number of initializations and the peculiarities of the training set on the generalization were investigated. The correlation between the optimized criterion (J_{mse} or ε_t) and the expected probability of error (ε_t) is suggested as quantitative measure of the phenomenon that is often called *over-training* in neural networks.

The relation between the classifier's complexity and the expected probability of error was investigated in *chapter 6*. A first problem that arose was how to quantify the complexity of a classifier. A measure proposed by Vapnik defines the capacity as the *maximum* number of points that can be given an arbitrary Boolean label. The approach that was taken in this chapter was to use the Foley capacity measure to analyse the capacity of feed-forward classifiers. This capacity measure is not based on a maximum capability but on an average classification capacity.

The main result of the Foley capacity approach is that the total number of parameters is not a good indication of the classifier's capacity. The capacity is *not* a linear function of the number of parameters if the number of hidden units is increased. It is furthermore observed that the number of iterations and a possible correlation in the data influences the classifier's capabilities.

The results of the capacity analysis were used to study the finite sample size properties and the relation between capacity and expected probability of error. It is shown here that the Foley capacity approach is valuable for analyzing the feed forward classifier and that small sample size properties agree with the conclusion of the Foley analysis.

In *chapter 7* real-world applications were used to investigate the usefulness of the feed forward network for pattern classifier. The NETtalk experiment was repeated and results comparable to [Sejnowski 1986] were obtained. An optimized nearest neighbor showed a better performance than the neural approach for this application. For the other six applications the nearest mean and nearest neighbor classifiers are better except for a marginally better performance of the neural classifier for the Iris data set.

A software library specially designed for neural network research was described in *chapter 8*. This simulation library is integrated with a statistical pattern recognition library and was used for all the experiments that were discussed in this thesis.

9·3 Discussion

The relation between classifier's complexity, data complexity, sample size and the expected probability of error is still an interesting topic for research in pattern recognition. The results reported in the neural network literature indicate that the traditional knowledge needed to be refined and this is confirmed in this thesis. The feed forward classifier is interesting because the complexity of this classifier is determined

by the number of hidden units in combination with the amount of training. This allows a controlled observation of generalization as function of the classifier's complexity.

In chapter 6 a data set was proposed (equation 6·7) to analyse the small sample properties of multi-layer perceptrons. From this equation four different data sets were generated by varying the statistics of the random variables that are present in this model. These sets differ in complexity and equation 6·7 allows one to generate pattern recognition problems for which the complexity can be controlled.

The small sample size problems have traditionally been studied for the specific Gaussian distribution with equal covariance structure. For the feed forward classifier and the proposed data set (equation 6·7) the small sample size problems can be studied for different data complexity and classifier complexity. The results of chapter 6 can serve as a starting point for further research of the effects of small sample sizes and how this is affected by varying the data complexity and classifier's complexity.

The Foley capacity analysis was proposed to allow a general statement about the number of free parameters and the classifier's complexity. The results of chapter 6 indicate that a ranking of expected probability of error can be predicted. It is an open question if the capacity curves can be used to predict the generalization properties of a classifier. Further research is needed to investigate if such an analysis can be used to formulate a better rule-of-thumb for pattern practitioners. The importance of a better definition or method to estimate the classifier's capacity has become clear in this thesis and is one of the most important issues for further research in pattern recognition.

The comparison of feed forward classifier discussed in chapter 7 indicated that a fair comparison is difficult. The performance of feed forward classifier is dependent on a large number of design issues and if these are optimized by using the test set this clearly leads to a too optimistic conclusion. The approach followed in chapter 7 probably leads to a pessimistic conclusion about the practical usefulness. Most design issues were ignored or chosen on an ad hoc basis in this comparison.

Although it has not become completely clear if the multi-layer perceptron is a more accurate classifier, the algorithm is simple to implement and a solution is obtained without much labor. It seems furthermore that feature extraction is less critical with this classifier because the hidden units perform some kind of feature extraction.

This thesis investigated a number of important properties of the neural classifier and some practical advantages and disadvantages have become clear. The extensive literature on the application of neural networks to real world problems, however, suggests that classifier is a valuable extension of the existing set of methods and algorithms.

References

[Amari 1992a] S. Amari, N. Murata and S. Yoshizawa, "Network information criterion, determining the number of hidden units for an artificial neural network model", Technical report, Department of Mathematical Engineering, University of Tokyo, Bunkyo-ku, Tokyo 113, Japan, June 1992.

[Amari 1992b] S. Amari and N. Fujita, "Four types of learning curves", Neural Computation, Vol 4, page 605-618, 1992.

[Anderson 1973] T.W. Anderson, "An asymptotic expansion of the distribution of the studentized classification statistic W", Ann. Stat., vol 1, page 964-972, 1973.

[Bakker 1993] R.R.N. Bakker, M.A. Kraaijveld, R.P.W. Duin and W.F. Schmidt, "On the speed of training networks with correlated features", Proceedings IEEE International Conference on Neural Networks, San Francisco, vol 2, page 919-922, April 1993.

[Baum 1989] E. Baum and D. Haussler, "What size net give valid generalization", Neural Computation, Vol 1, page 151-160, 1989.

[Boer 1991] P.J. den Boer, "Improving speed and performance of the back propagation algorithm", Masters Thesis, Faculty of Applied Physics, Delft University of Technology, Delft, The Netherlands, September 1991.

[Boullion 1975] T.L. Boullion, P.L. Odell and B.S. Duran, "Estimating the probability of misclassification and variable selection", Pattern Recognition, vol 7, page 139-145, 1975.

[Bowie 1977] J. Bowie and I. T. Young, "An analysis technique for biological shape II", Cytologica, vol 21, page 455-464, 1977.

[Bowker 1961] A.H. Bowker and R. Sitgreaves, "An asymptotic expansion for the distribution of the W-classification statistic", *Studies in Item Analysis and Prediction*, Solomon editor, page 292-310. Stanford University Press, 1961.

[Bryson 1969] A.E. Bryson and Y.C. Ho, *Applied Optimal Control*, New York Blaisdell, 1969.

[Buntine 1991], W.L. Buntine and A.S. Weigend, "Calculating second derivatives on feed forward networks", Technical Report, RIACS and NASA Ames Research Center, Moffet Field, CA, USA, 1991.

[Carpenter 1988] G.A. Carpenter and S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network", IEEE Computer, Special issue on Artificial Neural Systems, page 77-88, March 1988.

[Chandrasekaran 1977] B. Chandrasekaran and A.K. Jain, "Independence, measurement complexity and classification performance", IEEE Transactions on Systems, Man and Cybernetics, vol smc-5, no 2, page 240-567, July 1977.

[Cover 1965] T.M. Cover, "Geometrical and statistical properties of linear inequalities with applications in pattern recognition", IEEE Transactions on Electronic Computers, vol EC-14, page 326-334, June 1965.

[Cover 1974] T.M. Cover, "The best two independent measurements are not the two best", IEEE Transactions on Systems, Man and Cybernatics, Vol 7, page 657-661, 1974.

[Cox 1982] L.H. Cox, M.M. Johnson, K. Kafadar, "Exposition of statistical graphics technology", ASA Proceedings Statistical Computation Section, page 55-56, August 1982.

[Crick 1989] F. Crick, "The recent excitement about neural networks", Nature, volume 337, page 129-132, 12 Jan 1989.

[Day 1990] S.P. Day and D. Camporese, "A stochastic training technique for feed forward networks", Proceedings of International Joint Conference on Neural Networks, Vol 1, page 607-612, San Diego, 1990

[Deev 1972] A.D. Deev, "Asymptotic expansions of statistic distributions of discriminant analysis W, M, W*", Stat. Metody Klassif., MGU, Moscow, page 6-51, in Russian, 1972.

[Devijver 1982] P.A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*, Prentice-Hall International, London, 1982.

[Duda 1973] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, 1973.

[Duin 1976] R.P.W. Duin, "A sample size dependent error bound", Proceedings Third International Joint Conference on Pattern Recognition, Coronado, California, page 156-160, 1976.

[Duin 1978] R.P.W. Duin, *On the accuracy of statistical pattern recognizers*, Dutch Efficiency Bureau, Pijnacker, ISBN 90 6231 052 4, 1987.

[Efron 1973] B. Efron, "The efficientcy of logistic regression compared to normal discriminant analysis", Journal American Statistical Association, vol 70, page 892-898, 1975.

[Errington 1993] P.A. Errington and J. Graham, "Classification of chromosomes using a combination of neural networks", Proceedings of the 1993 International Conference on Neural Networks, page 1236-1241, San Francisco, April 1993.

[Fisher 1936] R.A. Fisher, "The use of multiple measurements in Taxonomic prblems", Ann. Eugenics, Vol 7, page 280-322, 1936.

[Foley 1972] D.H. Foley, "Considerations of sample and feature size", IEEE Transactions on Information Theory, vol it-18, no 5, September 1972.

[Fukunaga 1971] K. Fukunaga and D.R. Olsen, "An algorithm for finding intrinsic dimensionality of data", IEEE Transactions on Computers, Vol C-20, page 176-183, February 1971.

[Fukunaga 1984] K. Fukunaga and T.E. Flick, "An optimal global nearest neighbor metric", IEEE Transactions on Pattern Recognition and Machine Intelligence, Vol 6, No 3, May 1984.

[Gallant 1990] S.I. Gallant, "Perceptron-based learning algorithms", IEEE Transactions on Neural Networks, vol 1, no 2, page 179-191, june 1990.

[Gallinari 1991] P. Gallinari, S. Thiria, F. Badran and F. Fogelman-Soulie, "On the relationship between discriminant analysis and multi-layer perceptrons", Neural Networks, Vol 4, page 349-360, 1991.

[Gear 1971] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall Inc, Englewood Cliffs N.J., 1971.

[Goddard 1988] N.H. Goddard, K.J. Lynne and T. Mintz, "Rochester Connectionist Simulator", Technical Report 233, Computer Science Department, University of Rochester, Rochester, New York, March 1988.

[Golomb 1991] B.A. Golomb, D.T. Lawrence and T.J. Sejnowski, "Sexnet: A neural network identifies sex from human faces", Advances in Neural Information Processing Systems 3, Editors Lippmann, Moody and Touretzky, Morgan Kaufman Publishers, San Mateo, page 572-577, 1991.

[Golub 1989] G.H. Golub and C.F. van Loan, *Matrix Computations*, Second edition, Johns Hopkins University Press, 1989.

[Gorman 1988] R.P. Gorman and T.J. Sejnowski, "Learned classification of sonar targets using massively parallel network", IEEE Transaction on Acoustics Speech Signal Processing, Vol 36, No 7, page 1135-1140, July 1988.

[Guiasu 1977] S. Guiasu, *Information Theory with Applications*, McGraw-Hill International Book Company, 1977.

[Haas 1970] W. Haas, *Phonographic Translation*, Manchester University Press, Manchester, 1970.

[Haersma Buma 1976] C. Haersma Buma and R.P.W. Duin, "Computation of concave piecewise linear discriminant functions using Chebyshev polynomials", IEEE Transactions on Computers, Vol c-15 no 2, February 1976.

[Hampshire 1990], J.B. Hampshire II and B.L. Pearlmutter, "Equivalent proofs for multi-layer perceptron classifiers and the Bayesian discriminant function", Preprint for the Proceedings of the 1990 Connectionist Models Summer School, editors Touretzky, Elman, Sejnowski and Hinton, San Mateo, Morgan Kaufmann, 1990.

[Hartman 1990] E.J. Hartman, J.D. Keeler and J. M. Kowalski, "Layered neural networks with Gaussian hidden units as universal approximations", Neural Computation, Vol 2, page 210-215, 1990.

[Hebb 1949] D.O. Hebb, *The orginization of behavior*, New York: Wiley, 1949. Reprinted in J.A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of research*, Cambridge MIT press, 1988.

[Hertz 1991], J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Lecture Notes Volume I, Santa Fe Institute, Studies in the Science of Complexity, Addison Wesley publishing Company, 1991.

[Highleyman 1962] W.H. Highleyman, "Linear decision functions with applications to pattern recognition", Proceedings IRE-50, page 1501, 1962.

[Hoeffding 1963] W. Hoeffding, "Probability inequalities for sums of bounded random variables", American Statistical Association Journal, March 1963.

[Holler 1992] M. Holler, "80170NX: Neural network technology and applications", Intel publication 241359, Intel Corporation, 2200 Mission College Boulvard, Santa Clara C.A., USA, 1992.

[Hopfield 1985] J.J. Hopfield and D.W. Tank, "'Neural' computation of decisions in optimization problems", Biological Cybernetics 52, page 141-152, Springer Verlag, 1985.

[Jackobs 1988] R.A. Jacobs, "Increased rates of convergence through learning rate adaption", Neural Networks, volume 1, page 295-308, 1988.

[Jain 1978] A.K. Jain and W.G. Waller, "On the optimal number of features in the classification of multivariate Gaussian data", Pattern Recognition, vol 10, page 365-374, 1978.

[Jain 1982] A.K. Jain and B. Chandrasekaran, "Dimensionality and sample size considerations in pattern recognition practice", *Handbook of Statistics*, vol 2, page 835-855, North Holland Publishing Company, 1982.

[Jain 1987] A.K. Jain, R.C. Dubes and C.C. Chen, "Bootstrap techniques for error estimation", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 9, page 628-633, 1987.

[Jain 1988a] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ 1988.

[Jain 1988b] A.K. Jain and M.D. Ramaswami, "Classifier design with Parzen windows", *Pattern Recognition and Artificial Intelligence*, Gelsema and Kanal editors, Elsevier Science Publishers, page 211-228, 1988.

[Johansson 1990] E.M. Johansson, F.U. Dowla, D.M. Goodman, "Back propagation learning for multi-layer feed forward neural networks using the conjugate gradient method", Technical Report UCRL-JC-104850, Lawrence Livermore National Laboratory, September 1990.

[Kamata 1992] S. Kamata, R.O. Eason, A. Perez and E. Kawaguchi, "A neural network classifier for LANDSAT image data", Proceedings 11th International Conference on Pattern Recognition, Vol 2, page 573-576, September 1992.

[Kanal 1971] L. Kanal and B. Chandrasekaran, "On dimensionality and sample size in statistical pattern recognition", Pattern Recognition, vol 3, page 225-234, 1971.

[Kang 1993] K. Kang, J. Oh, C. Kwon and Y Park, Generalization in a two-layer neural network", Technical Report, Department of Physics, Pohang Institute of Science and Technology, Pohang, Kyongbuk, Korea, January 1993.

[Kernighan 1978] B. Kernighan and D. Richie, *The C programming language*, First edition, Prentice-Hall, 1978.

[Kharin 1984] Y.S. Kharin, "The invesigation of risk for statistical classifiers using minimum estimators", Theory Prob. Appl., vol 28, page 623-630, 1984.

[Kirkpatrick 1983] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, "Optimization by simulated annealing", Science, vol 220, page 671-680, Reprinted in J.A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, Cambridge MIT press, 1988.

[Kohonen 1982] Kohonen, T., "Clustering, Taxonomy, and Topological Maps of Patterns", Proceedings of the Sixth International Conference on Pattern Recognition, pp. 114-128, Munich, Germany, 1982.

[Kohonen 1988a] T. Kohonen, "The neural phonetic typewriter", IEEE Computer, Special issue on Artificial Neural Systems, page 11-22, March 1988.

[Kohonen 1988b] T. Kohonen, *Self-Orginization and Associative Memory*, Second Edition, Springer Verlag, 1988.

[Kohonen 1990] Kohonen, T., "The Self Organizing Map", Proceedings of the IEEE, Vol. 78, No. 9, pp. 1464 - 1480, September 1990.

[Kolen 1991] J.F. Kolen and J.B. Pollack, "Back propagation is sensitive to initial conditions", Advances in Neural Information Processing Systems 3, Editors Lippmann, Moody and Touretzky, Morgan Kaufman Publishers, San Mateo, page 860-866, 1991.

[Kolmogorov 1957] A.N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous of one variable and addition", Doklady Akademii Nauk USSR, Vol 114, page 953-956, 1957.

[Korn 1989], G.A. Korn, "A new environment for interactive neural networks", Neural Networks, Vol 2, No 3, 1989.

[Kraaijveld 1990] M.A. Kraaijveld and R.P.W. Duin, "On back propgation of edited data sets", Proceedings of the International Neural Network Conference, page 741, Paris, 1990.

[Kraaijveld 1993] M.A. Kraaijveld, *Small Sample Behavior of Multi-Layer Feedforward Network Classifiers: Theoretical and Practical Aspects*, Concept Ph.D. Thesis, Pattern Recognition Group, Department of Applied Physics, Delft University of Technology, Delft, April 1993.

[Krogh 1991] A. Krogh and J.A. Hertz, "A simple weight decay can improve generalization", Technical Report, The Niels Bohr Institute, Blegdamsvej 17, DK-2100 Copenhagen, Denmark, 1991.

[Le Cun 1986], Y. Le Cun, "Learning processes in a asymmetric threshold ntwork", *Disordered Systems and Biological Orginization*, editors Bienenstock, Fogelman Souli and Weisbuch, Springer Berlin, 1986.

[Le Cun 1989] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel, "Back propagation applied to handwritten zip code recognition", Neural Computation, Vol 1, page 541-551, MIT Press, 1989.

[Lee 1991a] D. Lee, S.N. Srihari and R. Gaborski, "Bayesian and neural network pattern recognition: a theoretical connection and empirical results with handwritten characters", *Artificial Neural Networks and Statistical Pattern Recognition, Old and New Connections*, editors Sethi and Jain, Elseviers Science Publishers, page 89-109, 1991.

[Lee 1991b] W.T. Lee and M.F. Tenorio, "On optimal adaptive classifier design criterion", Proceedings of the International Joint Conference on Neural Networks, Vol 2, page 385-389, Seattle, June 1991.

[Lehman 1988] A. von Lehman, E.G. Peak, P.F. Liao, A. Marrakchi and J.S. Patel, "Factors influencing learning by back-propagation", Proceedings IEEE International Conference on Neural Networks, vol 1, page 335-341, San Diego, 1988.

[Levelt 1993] D.F. Levelt, "NETtalk: a clever feed forward classifier of a simple calssification problem ?", Graduation Thesis, in Dutch, Faculty of Applied Physics, Delft University of Technology, Delft, The Netherlands, September 1993.

[Lipschutz 1991], S. Lipschutz, *Theory and problems of Linear Algebra*, second edition, Schaum's outline series, McGraw-Hill, 1991.

[Lorentz 1966] G.G. Lorentz, *Approximation of functions*, New York: Holt, Reinhart and Winston, 1966.

[Lou 1991] Z. Lou, "On the convergence of the LMS algorithm with adaptive learning rate for linear feed forward networks", Neural Computation, Vol 3, No 2, Summer 1991.

[Lowe 1991] D. Lowe and A.R. Webb, "Optimized feature extraction and the Bayes decision in feed forward classifier networks", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 13 no 4, April 1991.

[Lucassen 1984] J.M. Lucassen and R.L. Mercer, "An information theoretical approach to the automatic determination of phonemic baseforms", Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, page 42·5·1-42·5·4, 1984.

[Lyon 1988] R.F. Lyon and Mead, "An analog electric cochlea", IEEE Transaction on Acoustics Speech Signal Processing, page 1119-1135, July 1988.

[McCulloch 1943] W.S. McCulloch and W. Pitts, "A logical calculus of idea's immanent in nervous activity", Bulletin of Mathematical Biophysics 5, page 115-133, 1943. Reprinted in J.A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, Cambridge MIT press, 1988.

[Mead 1989] C. Mead, *Analog VLSI and Neural Systems*, Reading: Addison Wesley, 1989.

[Minsky 1969] M. Miskey and S. Papert, *Perceptrons: An introduction to computation geometry*, MIT press 1969.

[Moller 1990] M.F. Moller, "A scaled conjugate gradient algorithm for fast supervised learning", Technical Report PB-339, Computer Science Department, University of Aarhus, Denmark, November 1990.

[Mood 1974] A.M. Mood, F.A. Graybill and D.C. Boes, *Introduction to the Theory of Statistics*, Third edition, McGraw-Hill, 1974.

[Moody 1992] J. Moody, "The effective number of parameters, an analysis of generalization and regularization in non-linear systems", Advances in Neural Information Processing Systems 4, Editors Moody, Hanson, Lippmann, San Mateo, Morgan Kaufman Publishers, 1992.

[Murre 1992] J. Murre, "68 neurosimulators", Neuron Digest, Vol. 9, No. 15, March 1992.

[Natarajan 1991] B.K. Natarajan, *Machine Learning: A Theoretical Approach*, Morgan Kaufmann Publishers, San Mateo, 1991.

[Ness 1980] J. Van Ness, "On the dominance of non-parametric Bayes rule discriminant algorithms in high dimensions", Pattern Recognition, vol 12, page 355-368, 1980.

[NeuralWorks 1991] NeuralWorks Professional II/plus, *Reference Guide*, NeuralWare Inc., Penn Center West, building IV suite 227, Pittsburgh, PA 15276, USA, 1991.

[Nilsson 1965] N.J. Nilsson, *Learning Machines, Foundation of trainable pattern classifying systems*, McGraw-Hill Book Company, 1965.

[Okamota 1968] M. Okamoto, "Correction to an asymptotic expansion for the distribution of the linear discriminant function", Ann. Math. Statist., vol 39, page 1358-1359, 1968.

[Okamoto 1963] M. Okamoto, "An asymptotic expansion for the distribution of the linear discriminant function", Ann. Math. Statist., vol 34, page 1286-1301, 1963.

[Oppenheim 1983] A.V. Oppenheim, A.S. Willsky and I.T. Young, *Signals and systems*", Prentice-Hall International, 1983.

[Orfanidis 1990] S.J. Orfanidis, "Gram-Schmidt neural nets", Neural Computation, Vol 2, page 116-126, 1990.

[Owens 1989] A.J. Owens and D.L. Filkin, "Efficient training of the back propagation network by solving a system of stiff ordinary differential equations", Proceedings International Joint Conference on Neural Networks, Washingtion DC, page 381-389, June 1989.

[Parker 1985] D.B. Parker, "Learning logic", Technical report TR-47, Center for Computation Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA, 1985.

[Patrikar 1990] A. Patrikar and J. Provence, "Learning by local variation", Proceedings Intrnational Joint Conference on Neural Networks, Vol 1, page 700-703, Whashingtion D.C., 1990.

[Pettis 1979] K.W. Pettis, T.A. Bailey, A.K. Jain and R.C. Dubes, "An intrinsis dimensionality estimator from nearest neighbor information", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol PAMI-1, page 25-37, January 1979.

[Poggio 1989] T. Poggio and F. Girosi, "A theory of networks for approximation and learning", Massachusetts Institute of Technology, Artificial Intelligence Laboratory, A.I. memo 1140, C.B.I.P. Paper 31, July 1989.

[Pomerleau 1989] D. Pomerleau, ALVINN: An Autonomous Land Vehicle in a Neural Network. Advances in Neural Information Processing Systems, 1, Editor Touretzky, San Mateo, Morgan Kaufman Publishers, page 305-313, 1989.

[Press 1988] W. Press, B. Flannery, S Teukolsky and W. Vetterling, *Numerical Recipes in C*, Cambridge University Press, 1988.

[Raudys 1972] S. Raudys, "On the amount of priori information in designing the classification algorithm", Technical Cybernatics, vol 4, page 168-174, in Russian, 1972.

[Raudys 1976] S. Raudys, "On dimensionality, learning sample size, and complexity of classification algorithms", Proceedings Third International Joint Conference on Pattern Recognition, Coronado, California, page 166-169, 1976.

[Raudys 1980] S. Raudys and V. Pikelis, "On dimensionality, sample size, classification error and complexity of classification algorithm in pattern recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol pami-2, no 3, page 242- 252, May 1980.

[Raudys 1981] S. Raudys, "Influence of sample size on the accuracy of model selection in pattern recognition", Statistical Problems of Control, Vol 50, Inst. Math.& Cybernetics press, Vilnius, page 9-30, in Russian, 1981.

[Raudys 1982] S. Raudys and V. Pikelis, "Collective selection of the best version of a pattern recognition system", Pattern Recognition Letters 1, page 7-13, 1982.

[Raudys 1987] S. Raudys, "On accuracy of model selection in data analysis", Proceedings of III-rd International Conference on data Analysis and Informatics, INRIA Press, 91-98, 1987.

[Raudys 1991] S. Raudys and A.K. Jain, "Small sample problems in designing artificial neural networks", *Artificial Neural Networks and Statistical Patters Recognition, Old and New Connections*, editors Sethi and Jain , Elsevier Science Publishers B.V, page 33-50, 1991.

[Raudys 1992] S. Raudys, "Accuracy of feature selection and extraction in statistical and neural net pattern classification", Proceedings 11th International Conference on Pattern Recognition, volume B, page 62, September 1992.

[Richard 1991] M.D. Richard and R.P. Lippman, "Neural network classifiers estimate Bayesian *a posteriori* probabilities", Neural Computation, no 3, page 461-483, 1991.

[Ripley 1992] B.D. Ripley, "Neural networks and related methods for classification", Technical Report, Department of Statistics, University of Oxford, 1 South Parks Road, Oxford OX1 3TG, UK, Submitted to Royal Statistical Society Research Section, December 1992.

[Rosenblatt 1962] F. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, page 109, 1962.

[Ruck 1990] D.W. Ruck, S.K. Rogers, M. Kabrisky, M.E. Oxley and B.W. Sutter, "The multilayer perceptron as an approximation to a Bayes optimal discriminant function", IEEE Transactions on Neural Networks, vol 1, no 4, page 296-298, December 1990.

[Rumelhart 1986] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representation by error propagation", in Parallel Distributed Processing: *Explorations in the Microstructure of Cognition*, vol 1, editors Rumelhart and McClelland, Cambridge, MA, MIT Press, page 318-362, 1986. Reprinted in J.A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, Cambridge MIT press, page 673-695, 1988.

[Sabourin 1992] R. Sabourin and J.P. Drouhard, "Off-line signature verification using directional PDF and neural networks", Proceedings 11th International Conference on Pattern Recognition, Vol 2, page 321-325, September 1992.

[Sato 1991] A. Sato, K. Yamada, J. Tsukumo and T. Temma, "Neural network models for incremental learning", International Conference on Neural Networks, Helsinki, 1991.

[Schervish 1981] M.J. Schervish, "Asymptotic expansions of the means and variances of error rates", Biometrika, vol 68, page 295-299, 1981.

[Schmidt 1991] W.F. Schmidt, M.A. Kraaijveld and R.P.W. Duin, "A non-iterative method for training feed forward networks", Proceedings International Joint Conference on Neural Networks, Vol 2, page 19-24, Seatle, July 1991.

[Schmidt 1992] W.F. Schmidt, M.A. Kraaijveld, R.P.W. Duin, "Feed forward neural networks with random weights", Proceedings of the International Conference on Pattern Recognition, Vol II, page 1-4, The Hague, The Netherlands, September 1992.

[Schmidt 1993a] W.F. Schmidt and M.A. Kraaijveld, "ANNLIB - artificial neural network library and SPRLIB - statistical pattern recognition library", Introduction and Reference Manual, Technical Report Version 2·2, Pattern Recognition Group Delft, Department of Applied Physics, Delft University of Technology, Delft, The Netherlands, February 1993.

[Schmidt 1993b] W.F. Schmidt, S. Raudys, M. Skurikhina, M.A. Kraaijveld and R.P.W. Duin, "Initializations, back-propagation and generalization of feed forward classifiers", Proceedings IEEE International Conference on Neural Networks, Vol 1, page 598-604, San Francisco, April 1993.

[Sejnowski 1986] T.J. Sejnowski and C.R. Rosenberg, NETtalk: A parallel network that learns to read aloud. The Johns Hopkins University Electrical Engineering and Computer Science Technical Report JHU/EECS-86/01, 1986, Reprinted in J.A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, Cambridge MIT press, 1988.

[Sejnowski 1987] T.J. Sejnowski and C.R. Rosenberg, "Parallel networks that learn to pronounce English text", Complex Systems, Vol 1, page 145-168, 1987.

[Silva 1990] F.M. Silva and L.B. Almeida, "Accelerating techniques for the back propagation algorithm", Lecture notes in computer science, Almeida and Wellekens editors, Proceedings Eurasip Workshop 1990, Springer Verlag 1990.

[Sitgreaves 1973] R. Sitgreaves, "Some operating characteristics of linear discriminant functions", *Discriminant Analysis and Applications*, Cacoullos editor, page 365-374, Academic Press, 1973.

[Sjöberg 1991] J. Sjoberg and L. Ljung, "Overtraining, regularization and searching for a minimum in neural networks", Technical Report, Department of Electrical Engineering, Linkoping University, S-581 83 Linkoping, Swedem, 1991.

[Smith 1972] F.W. Smith, "Small sample optimality of design techniques for linear classifiers of Gaussian patterns", IEEE Transactions on Information Theory, vol it-18, no 1, january 1972.

[Stornetta 1987] W.S. Stornetta and B.A. Huberman, "An improved three layer back propagation algorithm", IEEE First International Conference on Neural Networks, San Diego, 1987.

[Tesauro 1990] G. Tesauro, "Neurogammon wins computer olympiad", Neural Computation, Vol 1, MIT Press, page 312-323, 1990.

[Trunk 1976] G.V. Trunk, "Statistical estimation of the intrinsic dimensionality of a noisy signal collection", IEEE Transactions on Computers, Vol C-25, page 165-171, February 1976.

[Tugay 1989] M.A. Tugay and Y. Tanik, "Properties of the momentum LMS algorithm", Signal Processing 18, page 117-127, 1989.

[Van den Bos 1982] A. van den Bos, "Parameter estimation", in P.H. Sydenham, *Handbook of Measurement Science*, vol 1, chapter 8, John Wiley and Sons Ltd., 1982.

[Vapnik 1982] V. Vapnik, *Estimation of dependeces based on emperical data*, Springer Verlag, 1982.

[Venezky 1970] R.L. Venezky, *The Structure of English Orthography*, Mouton, The Hague, The Netherlands, 1970.

[Waller 1978] W.G. Waller and A.K. Jain, "On the monotonicity of the performance of Bayesian classifiers", IEEE Transactions on Information Theory, 24, 392-394, 1978.

[Wan 1990] E. Wan, "Neural network classification: a Bayesian interpretation", IEEE Transactions on Neural Networks, vol 1, no 4, page 303-305, December 1990.

[Wasserman 1989] P.D. Wasserman, Neural Computation: Theory and Practice, Van Nostraand Reinhold, New York, 1989.

[Watrous 1987] R.L. Watrous, "Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization", Proceedings IEEE First International Conference on Neural Networks, San Diego, page 619-627, June 1987.

[Webb 1990] A.R. Webb and D. Lowe, "The optimized internal representation of multi-layer classifier networks perfroms nonlinear disciminant analysis", Neural Networks, Vol 3, page 367-375, 1990.

[Webster 1973] Webster's New College Dictionary, Editor B. Woolf, Merriam Webster, Merriam Company, Springfield, Massachusetts, USA, 1974.

[Weiss 1991] S.M. Weiss and C.A. Kulikowski, *Computer Systems That Learn*, Morgan Kaufmann Publishers, San Mateo, 1991.

[Werbos 1974] P.J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences", Ph.D. Thesis, Harvard University, 1974.

[Werbos 1991] P.J. Werbos, "Links between artificial neural networks and statistical pattern recognition", *Artificial Neural Networks and Statistical Pattern Recognition, Old and New Connections*, editors Sethi and Jain, Elseviers Science Publishers, page 11-31, 1991.

[Widrow 1960] B. Widrow and M.E. Hoff, "Adaptive switching circuits", 1960 IRE WESCON Convention Record, part 4, page 96-104, 1960, Reprinted in J.A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, Cambridge MIT press, 1988.

[Widrow 1985] B. Widrow and S Stearns, *Adaptive Signal Processing*, Prentice Hall, 1985.

[Wolf 1966] A.C. Wolf, "The estimation of the optimum linear decision function with a sequential random method", IEEE Transaction on Information Theory, volume IT-12, no 3, page 312-315, 1966.

[Wolfram 1992] S. Wolfram, *Mathematica: a System for Doing Mathematics by Computer*, Second Edition, Addison-Wesley Publishing Company, 1992.

[Wyman 1990] F.J. Wyman, D.M. Young and D.W. Turner, " A comparison of assymptotic error rate expansions for the sample linear discriminant function", Pattern Recognition, vol 23, no 7, page 775-783, 1990.

[Wyse 1980] N. Wyse, R. Dubes and A.K. Jain, "A critical evaluation of intrinsic dimensionality algorithms", *Pattern Recognition in Practice*, editors Gelsema and Kanal, North-Holland Publishing Company, page 415-425, 1980.

[Xu 1992] L. Xu, S. Klasa and A. Yuille, "Recent advances on techniques of static feed forward networks with supervised learning", Technical Report Harvard Robotics Laboratory, TR 92-2, Harvard University, Cambridge, 1992.

[Young 1974] I. T. Young, J. Walker and J. Bowie, "An analysis technique for biological shape I", Information and Control, vol 25, page 350-370, 1974.

[Young 1978] I.T. Young, "Further considerations of sample size and feature size", IEEE Transactions on Information Theory, vol it-24, no 6, page 773-775, Nov 1978.

[Young 1986] T. Y. Young and K.S. Fu, *Handbook of Pattern Recognition and Image Processing*, Academic Press, 1986.

Appendix A
Machine Learning Databases

In chapter 7 a number of public accessible data sets were used to evaluate feed forward classifiers against the nearest-mean and nearest neighbor classifiers. These data sets are available by *internet* using anonymous FTP. The machine addresses together with the directories were the data can be found are listed in table A·1. Use your e-mail as password and read the local instructions.

Table A·1: The hosts on internet with the public domain data sets that were used for the experiments of chapter 7.

Data	Machine	Account	Directory
NETtalk	pt.cs.cmu.edu	anonymous	/afs/cs/project/connect/bench
SONAR	pt.cs.cmu.edu	anonymous	/afs/cs/project/connect/bench
IRIS	ics.uci.edu	anonymous	/pub/machine-learning-databases
GLASS	ics.uci.edu	anonymous	/pub/machine-learning-databases
BLOOD	lib.stat.cmu.edu	statlib	/datasets

Samenvatting

In dit proefschrift wordt een belangrijke ontwikkeling op het gebied van neurale netwerken, *het meer-laagse perceptron*, behandeld vanuit een statistisch patroonherkennings standpunt. Het meer-laagse perceptron past heel goed in het raamwerk van een statistische classificator en is daarom gekozen als onderwerp van onderzoek. Een groot aantal toepassingen zijn er in de literatuur te vinden van het meer-laagse perceptron voor patroonherkenning en het is daarom interessant om deze methode te vergelijken met traditionele methoden.

Het vakgebied patroonherkennen is vooral gedomineerd door een statistische en mathematische aanpak en de traditionele methoden zijn daarom gebaseerd op mathematische en statistische argumenten. In neurale netwerken staan de menselijke hersenen en de neuronen centraal en die zijn de bron van inspiratie voor het meer-laagse perceptron geweest. Dit heeft duidelijk een ander model voor een patroonherkenned systeem opgeleverd. De argumenten die tot deze ontwikkeling hebben geleid zijn uitgelegd in dit proefschrift.

Het meer-laagse perceptron is onderzocht op theoretische en praktische aspecten. De aspecten die zijn onderzocht, zijn vooral die aspecten die in statistische patroonherkenning als belangrijk zijn ervaren in het verleden. De volgende lijst van onderwerpen is aan de orde geweest:

- De capaciteit of complexiteit van het meer-laagse perceptron.
- Het bepalen van de onbekende parameters van het meer-laagse perceptron.
- De asymptotische eigenschappen van het meer-laagse perceptron.
- De gevolgen van het eindige aantal voorbeelden in de leerverzameling.
- De relatie tussen de complexiteit en de verwachte foutkans.
- Het selecteren van de beste classificator uit een aantal alternatieven.

De toepasbaarheid van het meer-laagse perceptron is onderzocht door deze te vergelijken met twee traditionele classificatoren voor zeven echte patroonherkennings problemen. De naaste-nabuur en naaste-gemiddelde classificatoren zijn gebruikt in deze vergelijking omdat deze resultaten ook iets zeggen over de structuur van het probleem. Deze klassieke methoden zijn beter dan de resultaten die er met het meer-laagse perceptron zijn gemeten. Deze vergelijking is waarschijnlijk pessimistisch omdat een eerlijk vergelijk problematisch is.

De vraag of het voordeel heeft om het meer-laagse perceptron te gebruiken is moeilijk te beantwoorden. De nauwkeurigheid is misschien niet optimaal maar deze methode lijkt minder afhankelijk van een goede keuze van de kenmerken die er worden geselecteerd voor de herkenning. In dit proefschrift wordt het aannemelijk gemaakt dat de zogenaamde verborgen neuronen in dit model, een soort kenmerk selectie uitvoeren.

Dit is een mogelijke verklaring waarom kenmerk selectie iets minder kritisch is dan bij traditionele methoden, waarbij deze twee fasen bij het ontwerpen vaak gescheiden zijn.

Verder is een belangrijke conclusie uit dit proefschrift dat het aantal leer-objecten dat er nodig is voor het optimaliseren van het meer-laagse perceptron, beduidend minder is dan wat er op grond van de traditionele kennis te verwachten was. Dit geeft duidelijk aan dat de klassieke vuistregels te pessimistisch zijn en dat dit een punt van nieuw onderzoek is. In dit proefschrift is een mogelijke kwantificering van de complexiteit van een classificator voorgesteld die mogelijk als een alternatieve definitie van de Vapnik-Chervonenkis capaciteit kan dienen voor vervolg onderzoek.

Dankwoord

Het tot stand komen van mijn proefschrift zou zonder ondersteuning niet mogelijk zijn geweest. Ik ben daarom iedereen die hieraan heeft bijgedragen veel dank verschuldigd.

Ik ben in het bijzonder veel verschuldigd aan *Liesbeth* die vooral het laaste jaar van mijn promotie veel heeft moeten opofferen. Ook voor haar was het een spannend en vermoeiend jaar en ik ben in die tijd niet altijd een even grote steun geweest voor haar als zij voor mij.

Verder wil ik mijn *ouders* bedanken voor hun interesse en aanmoedigingen om promovendus te worden. De vele *'vergissingen van de bank in mijn voordeel'* hebben zeker geholpen om mijn promotie goed af te ronden. Van deze financiele ondersteuningen heb ik niet alleen mijn Macintosh computer kunnen aanschaffen, maar ook de promotie kosten kunnen betalen.

Naast mijn ouders wil ik ook *Boudewijn* en *Heleen* bedanken voor hun aandacht voor mij en mijn werk.

Op mijn werk heb ik vele uren gediscuseerd met *Martin Kraaijveld*, die altijd klaar stond om te praten over mijn nieuwe ideen. Deze discussies heb ik altijd zeer leerzaam gevonden maar waren daarnaast ook erg gezellig. Ik vond het dan ook jammer dat jij vertrok.

De weinige ideen die de kritische discussies met Martin overleefden werden nog eens getoest aan *Bob Duin* zijn oordeel. Hij heeft als mijn co-promotor met zijn grote ervaring en kennis de grote lijnen in de gaten gehouden. Ook alle discussies die wij hebben gevoerd heb ik altijd zeer leerzaam en stimulerend gevonden.

Verder wil ik *Ted Young* bedanken voor zijn inzet voor de groep en die in de laaste fase van mijn werk mijn concept nog eens kritisch heeft bekeken. Ik ben zeer onder de indruk van de snelheid waarmee je dit hebt gedaan. Als laaste wil ik *Aarnoud Hoekstra* bedanken voor de vele *'puntjes op de i'* die hij nog uit mijn concept heeft gehaald.

De overige stafleden *Pieter Jonker*, *Piet Verbeek* en *Albert Vossepoel* wil bedanken voor hun zorg voor onze sectie en al hun op en aanmerkingen die mijn werk in de loop van de tijd hebben gestuurd. Speciaal Pieter Jonker wil ik bedanken voor het feit dat hij mij heeft betrokken bij DIAC. *Ad Herweijer* bedankt voor het regelen en bijhouden van mijn 5% KSO inzet.

Voor de goede technische ondersteuning, die misschien niet zichtbaar is in dit proefschrift, maar die zeker zo belangrijk is, wil ik *Rob Ekkers*, *Tom Hoeksma*, *Wim van Oel*, *Wouter Smaal* en *Jan Straver* bedanken. De altijd vrolijke en behulpzame secretaresses *Jose de Bruin* en *Annelies Frijters* bedankt voor jullie inzet.

De collega AIO/OIO's wil ik niet vergeten. *Henri Vrooman* voor zijn bridge tips, *Ben Verwer* voor onze korte DIAC samenwerking, *Jim Mullikin* voor zijn eigen gebrouwen bier, *Fons Verbeek* voor het zeilen, *Hans Netten* voor de uurtjes in TP-Cafe, *Hans Buurman* voor zijn UNIX tips, *Karel Straster* voor het introduceren van alcohol vrij bier, *Frank Boddeke* voor zijn organisatorische inzet, *Erik Bouts* voor zijn grappen en grollen, *Carol Orange* voor haar systeem beheer en last but not least *Lucas van Vliet*, sorry voor het slechte bridgen de laaste tijd.

De ontwikkeling van *SRPLIB* en *ANNLIB* is mede door de bijdragen en opmerkingen van *Chris Nieuwenhuize*, *Bart Venlet*, *Daan van Setten* en *Robert Bakker* geworden wat het nu is. Het onderzoek van hoofdstuk 7 met betrekking tot de NETtalk experimenten is het resultaat van een plezierige samenwerking met *David Levelt*.

Verder wil ik het *Universiteits-fonds* en het *VSB-fonds* bedanken voor hun ondersteuning van mijn bezoek aan de groep van dr. Anil Jain, Michigan State University, East Lansing, USA. Verder wil ik *Shell Nederland B.V.* en *NWO* (SIR-beurs) bedanken voor hun financiele ondersteuningen van mijn conferentie bezoeken.

I would like to thank *dr. Anil Jain* for giving me the opportunity of visiting his pattern recognition and image processing laboratorium. During my visit *Jianchang Mao* was of great help in finding my way around and I am still embarred by his hospitality. Best wishes to you, *Yao* and *David* and I hope to meet you in the future.